FACULTY OF ENGINEERING OF THE UNIVERSITY OF PORTO

Performance Analysis of a Database Caching System In a Grid Environment

Luis Albino Nogueira Ramos

Graduated in Informatics and Computing Engineering
By the Faculty of Engineering of the University of Porto

Dissertation submitted in partial fulfillment of
the requirements for the degree of
Master of Informatics Engineering

Dissertation prepared under the supervision of
Dr. Gabriel de Sousa Torcato David
from the Department of Electrical and Computer Engineering
of the Faculty of Engineering of the University of Porto

Porto, September 2007

to little birds who are the secrets of living

Resumo

O CERN é o maior centro de investigação em física de partículas do mundo onde milhares de físicos estudam os constituintes fundamentais da matéria. O maior projecto do CERN é o Large Hadron Collider (LHC), um acelerador de partículas constituído por vários detectores que observam e registam as colisões de partículas que acontecem no LHC.

A LHC Computing Grid (LCG) fornece à comunidade de físicos uma infra-estrutura de computação distribuída para armazenamento, distribuição e análise dos 15 PB de dados produzidos por ano pelo LHC. A LCG fornece serviços e infra-estruturas para a replicação e acesso distribuído a dados em ficheiros. Dado que parte dos dados são armazenados em bases de dados, serviços e infra-estruturas similares são necessários para esses dados.

O projecto Distributed Deployment of Databases (LCG3D) fornece a infra-estrutura de distribuição de bases de dados da LCG e é responsável pela investigação de técnicas de distribuição de bases de dados a usar na LCG. As técnicas de distribuição de bases de dados usadas são: a solução comercial para replicação de bases de dados Oracle Streams, e o sistema de cache de bases de dados FroNtier. FroNtier é uma técnica de distribuição de bases de dados baseada no caching dos resultados de queries. Uma hierarquia distribuída de caches é colocada entre os clientes da base de dados e o servidor de base de dados.

Esta tese propõe, em estreita colaboração com o projecto LCG3D, a elaboração de uma análise sistemática de desempenho do pacote de software FroNtier. Adicionalmente, a tese documenta os problemas de consistência da cache em discussão na comunidade FroNtier.

Inicialmente, esta tese documenta o estudo das áreas de investigação e tecnologias envolvidas: computação em grid, técnicas de distribuição de bases de dados, análises de desempenho e monitorização de desempenho.

A metodologia da análise consiste na definição de um conjunto de cargas de trabalho, factores, métricas, e num conjunto de experiências que melhor combinam estes valores.

O principal produto resultante desta tese é uma plataforma de testes de desempenho baseada em Linux desenvolvida para automação da execução das experiências da análise.

Os resultados mais relevantes da análise de desempenho ao FroNtier são: (i) a compressão de dados no servidor FroNtier melhora o desempenho global do sistema; (ii) o nível 1 é o melhor nível de compressão; (iii) um servidor FroNtier serve queries de 20-30MB sem problemas e até 80 clientes simultâneos; (iv) testes executados com uma carga de trabalho real mostram que um sistema Oracle tem um desempenho melhor que um sistema FroNtier. No entanto, a resolução dos problemas detectados no FroNtier, juntamente com a utilização de caches espalhadas pela grid inverterão este resultado; (v) genericamente, uma instalação FroNtier cobrindo uma base de dados permite que mais clientes sejam suportados.

As soluções para os problemas de consistência das caches são apresentados sob a forma de políticas para a manutenção da consistência de dados e, numa abordagem diferente, é apresentado um protótipo de um mecanismo de invalidação de cache.

O sistema FroNtier tem ainda de provar a sua fiabilidade e o seu desempenho. Apesar de ser muito atractivo como uma solução leve para a distribuição de bases de dados, os problemas de consistência de dados e a falta de provas de desempenho em cenários reais são ainda factores que afastam os interessados.

Abstract

CERN is the largest particle physics centre in the world where thousands of physicists study the fundamental constituents of matter. CERN's latest project is the Large Hadron Collider (LHC), a particle accelerator containing several detectors that observe and record the particle collisions happening in the LHC.

The LHC Computing Grid (LCG) provides the physics community with a distributed computing infrastructure for storage, distribution, and analysis of the 15 PB of data produced per year by the LHC. LCG provides with services and infrastructures for the replication and distributed access to file based data. As part of the data will be stored in databases, similar services and infrastructures are required for data stored in relational databases.

The project entitled Distributed Deployment of Databases (LCG3D) provides the database distribution infrastructure for the LCG and is responsible for the investigation of database distribution techniques to be used in the LCG. The database distribution techniques used are the commercial product for database replication Oracle Streams, and the database caching system FroNtier. FroNtier is a database distribution technique based on the caching of database query results. A distributed hierarchy of cache servers is introduced between the database clients and the database server.

This thesis proposes, in close collaboration with the LCG3D project, the elaboration of a systematic performance analysis of the FroNtier software package. Additionally, this thesis documents the cache consistency issues under discussion inside the FroNtier community.

Before digging into the FroNtier performance analysis, this thesis documents an extensive study of the research areas and technologies involved: grid computing, database distribution techniques, performance analysis, and performance monitoring.

The analysis methodology consists in defining a set of workloads, factors and performance metrics together with designing a list of experiments that best combine these values.

The main deliverable result of this thesis is a Linux based performance test framework developed to automate the execution of the analysis' experiments.

The FroNtier performance analysis most relevant results are: (i) payload compression on the FroNtier server boosts the systems' global performance; (ii) level 1 is the best compression level; (iii) FroNtier can handle queries of 20-30MB with no issues and up to 80 clients working simultaneously; (iv) tests performed with a real scenario workload showed that Oracle performs better than the FroNtier setup. Nevertheless, the resolution of the issues detected in the FroNtier setup together with the use of Squid caches spread around the grid will invert this result; (v) generally, a FroNtier setup covering an Oracle database allows a much larger number of clients to be handled.

The available solutions to the cache consistency issues are presented as cache consistency policies, and on a different approach, a cache invalidation mechanism is prototyped.

FroNtier has still to prove its reliability and performance. Despite being very attractive as a light-weight solution for database distribution, cache consistency issues and lack of real scenario performance proofs are still setting stakeholders aside.

Table of Contents

Table of Figures

Chapter 1 - Introduction

This chapter presents some background information about the institution where this project took place, and how the work developed for this thesis fits in the institution's projects. The thesis motivations and objectives are stated, and finally, the structure of the thesis is described.

1.1 Background

This section describes the scientific background in which the project was developed, namely, the institution, the particle accelerator, the scientific experiments underway, the computing challenges that arise from particle physics, and the computing grid.

1.1.1 CERN

The European Organization for Nuclear Research (CERN) [1] is the largest particle physics centre in the world. It is situated west of Geneva between France and Switzerland and it is currently funded by 20 European countries. In 1954, CERN started the European research on High Energy Physics (HEP). HEP studies the fundamental constituents of matter and forces between them (quarks, electrons, neutrinos, photons, etc.). Higher and higher energies are required to delve deeper and deeper into matter.

CERN's greatest scientific achievements are the discovery of the neutral currents in 1973, and the discovery of the W and Z bosons in 1983. These discoveries together confirmed the electroweak theory of the Standard Model that unified the electromagnetic force and the weak force. For these discoveries and others, several CERN collaborators were awarded the Nobel Prize in Physics.

Cutting-edge particle physics research requires advanced technology and CERN works closely with the industry. The applications of particle research done at CERN are vast and include fields as medicine, technology, industry, and research in other fields.

CERN's biggest technological achievement with a world wide impact in culture, economy, politics, and everyday life happened in 1989 when Tim Berners-Lee, a scientist at CERN, conceived and developed the World Wide Web for automatic information sharing between scientists working in different universities and institutes all over the world.

At the bottom line, in its quest for higher particle interaction energies, CERN exists primarily to provide European physicists with tools that meet research demands at the limits of human knowledge, namely particle accelerators and detectors.

1.1.2 Large Hadron Collider (LHC)

Presently, the main research program at CERN is the particle accelerator Large Hadron Collider (LHC) [2] approved in 1995 and planned to start operations in 2007 with a final cost of almost 3000 million euros. It lies in a circular tunnel around 100 meters below the ground with a circumference of 27 kilometers. LHC will generate two beams moving in opposite directions inside the tunnel with particles traveling at 0.035km/h below the speed of light. LHC is a versatile accelerator as it can collide beams of both proton and heavy ions.

One of the main innovations of the LHC is its size and energy level. LHC will heat matter at a temperature 100000 times higher than the temperature at the center of the sun, and will compress it to a point that the pyramid of Kheops would fit in a pinhead. When operating, LHC will be the world's largest and highest energy particle accelerator. LHC will enable the discovery of new physics and dig deep into matter. One of the expected scientific results coming from LHC is the observation of the Higgs boson that is the predicted by the electroweak theory, and is the only Standard Model particle not yet observed.

LHC have in different points of the tunnel four experiments where particle beams will collide. These experiments are huge machines that incorporate hundreds of different detectors that will observe and record what happens when the beams collide. LHC and its experiments will run and produce huge data amounts during approximately 20 years. It is expected a data volume of 15 PB (15360 TB) per year. This data will be accessed and analyzed by half of the world's particle physicists (around 5000) in some 500 research institutes and universities from more than 80 countries worldwide.

1.1.3 LHC experiments

The four LHC experiments are A Toroidal LHC ApparatuS (ATLAS), the Compact Muon Solenoid (CMS), the LHC-beauty (LHCb) and A Large Ion Collider Experiment (ALICE).

LHCb is a specific experiment designed to study the decay of a type of particles called B-mesons.

ALICE is a general-purpose experiment optimized for heavy-ion reactions that aims to study the physics of the quark-gluon plasma: a phase of matter formed at very high energy densities.

CMS and ATLAS are general purpose proton-proton detectors designed to exploit the full discovery potential of the high energy beams generated by the LHC, for example, the observation of the Higgs particle, which explains the origin of the spontaneous symmetry breaking mechanism in the electroweak sector of the Standard Model.

ATLAS will be the largest and more complex detector ever built with 45 meters long and 25 meters in diameter. CMS is the heaviest detector complex weighting more than 10 thousand tones.

The LHC will produce a beam of particles that collide 40 million times per second in each experiment's pit. Here, each one of these collisions (also called events) produces thousands of particles that are tracked by detector complexes deployed around the main accelerator tube in a layered structure.

These huge assemblies of measurement devices can identify the position of all types of particles such as hadrons, electrons, photons, and muons to a fraction of a millimeter and distinguish its individual track among thousands of other particles.

To interact and read from all these devices a very large set of electronic and computational components have to be deployed. Depending on the experiment the names and roles of these systems may change. Here, a generic overview is made. Apart from the low level electronics that constitute all detectors, the typical components of a detector system are a Data Acquisition System, a High-Level Trigger system, and a Control System.

1.1.3.1 Data Acquisition System (DAQ)

The task of a DAQ system is to control the complete data acquisition process that consists of gathering information about the collisions (events), analyzing them, selecting the most

interesting ones, collecting all detector information at the time of each event, and storing them for further off-line analysis.

The process starts at the lowest level components of a detector: the detector's read out electronics that identify and measure the physical events. The main functions of a DAQ system are to realize the dataflow from these detector electronics up to the data storage. The major steps in this process are the assemblage of all event fragments gathered by different hardware components (event building) and the filtering of events to meet storage availability (event filtering) before writing selected events to mass storage.

At the LHC, bunches of particles will collide 40 times per second and each of these crossings will result in an average of 20 proton-proton events. The data size of one of these events is fixed and is defined by each experiment based on its requirements: the event sizes go from 25kB in LHCb to 12,5 MB in ALICE. The very high event rates (almost 1Ghz of proton-proton events) together with the referred event sizes will result in very high data rates from the detector's electronics to the DAQ system. If no filtering is done, this operation would demand a storage capacity many orders of magnitude larger than the available nowadays.

Through the steps of data flow management, the DAQ system is thus responsible for reducing the data rates so that the mass storage available is efficiently used. This event filtering process is a rather important step for the overall success of the scientific task as from the initial 40 million bunch collisions per second only about 100 are stored for analysis (a reduction of O(10^7)).

This selection process is usually too complex and its rejection rate to high to be achieved in only one pass, so it is done in several steps (trigger levels) that are a mix of hardware and software triggers. Each trigger level has an average acceptance rate that depends on the amount of the data needed to store the information about an event (event size), the frequency of events (event rate), the available bandwidth between system resources, etc.

The DAQ system is thus not only responsible for getting the detector data by interacting with the detector electronics, but also for interacting with the different trigger systems that select the most interesting events.

The first data selection is done directly after the measurement at the electronics level where fragments of an event are detected and analyzed.

The first level triggers are the first software triggers that are typically a event-filter CPU-farm that makes a short first analysis on given event fragments and decides whether the event fragment should be stored for further analysis or not.

Upon a positive first level trigger decision about an event, the fragments belonging to a specific event are assembled (event building) and sent to the next event filtering stages. This process ends with the High-Level Trigger that is the final event filtering stage. It is the responsibility of the DAQ to control the data flow to and from the High-Level Trigger farm.

After filtering and assembling all information about a given event the DAQ system is responsible, as a final stage, for writing the accepted events to permanent data storage for further analysis.

The DAQ system also performs data quality and performance monitoring and overall control of the system.

Though there are a lot of commonalities between experiments, a DAQ system has to be heavily adapted for each experiment.

1.1.3.2 High-Level Trigger system

The High-Level Trigger system (HLT) is the last filtering and more elaborated stage before a given event goes to storage and further off-line analysis.

The HLT algorithms operate on full information gathered for a specific collision (event data) plus the detector configurations at the time of the event, and are in some cases responsible for a first event classification.

The task of the HLT is to select the most relevant events by executing physics selection algorithms on a large collection in order to reduce the data volume. This should be done preserving the events with the most interesting physics content.

These objectives are achieved by using a rather large computer cluster where the event rate is reduced by detailed analysis of its physics. The event size is reduced by keeping only the interesting parts of a given event and by compressing event data.

The output rate of the HLT filtering is scaled to the event size of the experiment. So, for example, ALICE events will have an output rate of 100hz (HLT will classify as interesting 100 events per second in average), while LHCb events will have an output rate of 2000hz. This means that the throughput needed to move the data from the HLT to the storage can reach values of 1,25GB per second (100 events of 12,5 MB each, per second). Event derived data should also be taken into account on these calculations as, for example, the reconstruction data size per event is around 100kB per event.

1.1.3.3 Control System

In order to make human interaction with such a machine as an experiment possible, one integrated system with coherent user interfaces and system responses should be deployed. The control system of an experiment provides a unified view of the experiment and a central point from where all operations are initiated and controlled, typically a control room. It also provides a collection of interfaces that allow independent concurrent activities on parts of the experiment by different operators, for example, non-expert shift-crews and sub-detector experts.

The control system acts as glue between the various elements of the experiment as the DAQ, the HLT, and all other experiment equipment. It ensures the correct and safe operation of the experiment equipment providing control, configuration, and monitoring functionalities. Depending on the experiment, this system is structured in different sub-systems.

The ultimate objective of such a system is to assure the high quality of the physics data taken by the experiment. Moreover, it is supposed to be operational throughout all phases of the experiment thus having strong requirements on availability and reliability.

Examples of tasks achievable through an experiment control system are the general basic control, configuration and monitoring of different systems as the detector electronics, the event-filter farm, the cooling systems, the power supplies, etc. This implies that the control system is able to communicate and coordinate operations with these specific lower level components and control systems. The control system is also the interface with all external infrastructural systems and services as the LHC accelerator itself or the global safety system.

The control system usually includes functionalities to control, configure, and monitor the DAQ system, but excludes any management, processing, or transportation of event data. The DAQ system deals with event data read from the detectors, while the control system deals with data related to the operational system status and parameters of the machine when the

event data was taken. Close interaction between these systems is required as these two types of data need to be correlated for offline analysis.

In terms of monitoring services, the control system can display available online status information and operational configuration of all the experiment equipment and can also signal any abnormal behavior like errors, warnings, alarms, or just diagnostics messages sent by different applications.

In terms of configuration services, a control system provides a framework for storing and accessing large amounts of information describing the online system.

In terms of control, the system handles all actions initiated by the operator, gives guidance to the operator, implements verification, diagnostic and recovery mechanisms such as being capable of taking appropriate actions automatically in specific situations.

1.1.4 Worldwide LHC Computing Grid (LCG)

All four experiments have different computing models for the analysis of the data after the described process of data acquisition. Nevertheless, all experiments have the same goal: to distribute the huge amounts of data taken down in the experiment pit to hundreds of institutes around the world, and provide the physics community with a computing resource for further analysis.

A traditional approach to this problem would be to centralize all the needed storage and computing power at one location near the experiments. For the LHC, a distributed approach was chosen as it offers two main advantages, namely the distribution of the significant costs of maintaining and upgrading the needed resources, and the absence of single points of failure with data replication and computing resources failover. On the other hand, a distributed system like a grid presents significant challenges as ensuring network bandwidth between resources, managing software versions installed in different sites, managing and protecting the data in heterogeneous environments, etc.

To face this challenge the project Worldwide LHC Computing Grid (LCG) [3], [4] was setup in 2001. The LCG project has the mission to develop, build and maintain a distributed computing infrastructure for the storage and analysis of data from the four LHC experiments for the entire HEP community that will use the LHC.

LCG is a collaboration between CERN (the host laboratory) with the four LHC experiments and all the computer centers around the world that commit to provide resources for the LHC data storage and analysis. LCG represents an enormous technological challenge that will assure the access to the proposed computing infrastructure provided by more than 100 institutions in more than 20 countries and involving more than 25 funding agencies.

In order to plan the deployment of the LCG infrastructure, the experiments have put together a number of requirements for the LCG. These requirements drive the LCG resources needs and software developments. Though each experiment has defined quite different requirements in terms of data volumes, computing power and services a lot of commonalities exist. In terms of total data volumes, the experiments require a CPU capacity of around 100.000 CPUs, about 60 PB of disk storage and 50 PB of mass storage.

The infrastructure that LCG will provide to support the computing models and requirements of the LHC experiments is a computing grid that will rely on a set of software packages commonly called grid middleware. These packages offer grid users a set of functionalities such as authentication, job submission, distributed file management, storage services, and distributed database services.

The LCG is implemented in a distributed tiered model:

- Tier-0 is the accelerator centre at CERN which is divided in:
  - the online layer: the experiments' pits where data acquisition is done);
  - the offline layer where data is backed up, initially processed, and then distributed to Tier-1 centers.
- Tier-1 centers are 11 large centers spread around the world with sufficient storage capacity for a large fraction of the data that will hold most of the data analysis.
- Tier-2 sites are around 100 centers in around 40 countries that can store sufficient data and provide adequate computing power for simulation and end-user data analysis.
- Tier-N computing resources are mainly local clusters in university departments or individual PCs used by individual scientists to access grid facilities.

1.1.5 Data distribution, processing and analysis

LHC data will flow from online systems in the experiments' pits to hundreds of points around the world where it will be managed, processed and analyzed. This distribution will follow the hierarchical architecture of the grid.

Depending on the experiment, the first processing of data can be done either on the online farm of the experiment or at the Tier-0 farm. In all cases a master copy of the data is stored on tape at Tier-0, and a second copy of the data is stored across the Tier-1 centers associated with the experiment.

Each Tier-1 centre serves a number of LHC experiments (most of them serve several experiments). The responsibilities of Tier-1 centers depend on the experiment. Typically, Tier-1s are responsible for managing their data storage systems and for providing computing power. These resources will be used by re-processing and analysis processes.

Tier-2 centers will provide computational power and storage services for simulation and for end-user analysis. Tier-2 centers obtain data from Tier-1 centers and send generated data back to Tier-1 centers for permanent storage.

Tier-N facilities lying in universities and laboratories will also process and analyze the LHC data.

When a given event passes all the online filtering stages, it is sent for off-line processing where new higher level data is generated from it. Event data gathered directly in the detector, like times and voltages, are referred as raw data. This data has to be processed several times before being analyzed by physicist. Typical off-line data processing tasks are calibration, alignment and reconstruction.

Alignment and calibration are procedures tasks that, from raw data, generate non-event data that is needed for the reconstruction of event data as, for example, information about the detector's configuration at the event time. These tasks are performed several times during the event lifecycle from triggering to analysis.

Reconstruction is the process where the raw data is reconstructed into tracks and energy levels. This is generally a CPU-intensive programmatic activity that requires an extended period of time. Reconstruction jobs need access to condition and calibration data relevant to the particular raw data under reconstruction. The final output of a reconstruction process is referred as reconstructed data.

For each event, reconstruction can be done several times over raw data and over already reconstructed data. Moreover, further processing (alignment and calibration) of reconstructed

data is performed and leads to improved calibration data that can be again reconstructed. At any stage, end user analysis can be performed using reconstructed data.

These processing steps vary between experiments but they always produce different data formats describing the event, each of them providing different levels of detail. All different types of data, from raw to processed data types, have to be managed and stored in permanent data storage either in Tier-0 and/or Tier-1s. Thus, raw data will be processed (alignment and calibration) and reconstructed at the Tier-0 according to the scheme of the experiment, and the resulting datasets will be distributed to Tier-1 sites. These processing stages continue with further event reconstruction taking place at Tier-1s. Selected sets of event data are transferred to Tier-2 centers which support iterative analysis by users. As physics applications may need to navigate from reconstructed data to raw data, usually, raw events and corresponding reconstructed data are stored in the same site.

1.1.6 Summary

"Discovering new fundamental particles and analyzing their properties with the LHC accelerator is possible only through statistical analysis of the massive amounts of data gathered by the LHC detectors ATLAS, CMS, ALICE and LHCb, and detailed comparison with compute-intensive theoretical simulations." [3].

This generic problem statement is a rather typical one in HEP computing. In HEP computing most of the data is read-only, there is a small number of concurrent users, the transaction rates are very low, data loss is acceptable (tolerance to certain degrees of data inconsistency), and there are only a few data dependencies as each event is processed independently. On the other hand, HEP computing involves huge data volumes and rates (up to 100PB, 1.5GB/s), fully distributed environments, and project lifetimes of several years with the continuous development of new analysis applications (around 20 years for the LCG).

With this background context, this thesis will describe how grid and database technologies are used to solve these computing problems.

1.2 Motivation

The accelerator LHC will produce huge amounts of data that will be distributed and analyzed using the LCG. LCG provides services and an infrastructure for replication and distributed access to file based data. Physics applications and grid services require a similar infrastructure for data stored in relational databases as several applications and services already use RDBMS.

The project Distributed Deployment of Databases (LCG3D) [5] was setup to provide the database distribution infrastructure for the LCG. This includes the coordination of all the database deployments in the LCG and the investigation of database distribution techniques to use in the LCG.

The database distribution techniques used in the LCG3D project are the commercial product for database replication Oracle Streams, and the database caching system FroNtier. Both these offer advantages and disadvantages. Extensive tests of these technologies are taking place at CERN in order to validate if they fulfill the LCG database requirements. Inspired by these activities, this thesis was developed, in collaboration with the LCG3D project, based on the validation tests of the FroNtier package, more specifically, the evaluation of its performance.

Additionally, several issues has been raised inside the community using FroNtier as whether the cache consistency offered by the FroNtier package suits the applications that are using it. This discussion gave origin to a study on data consistency issues and the definition of cache consistency policies, and on a different approach, the development of a cache invalidation mechanism that would solve the consistency issues.

1.3 Objectives

The main objective of the thesis is to elaborate a systematic performance analysis of the database caching solution offered by the FroNtier package. This analysis should include extensive performance testing of FroNtier which results in a FroNtier benchmark suite. This benchmark should allow one to reproduce the analysis of any FroNtier servers, and help estimating setup configurations for given performance/deployment needs.

The FroNtier performance evaluation together with the output of other test activities in the LCG3D project will enable an interesting comparative analysis between FroNtier and other database distribution techniques. This comparative analysis should be performance oriented and FroNtier centered. Among the alternatives to FroNtier, a special focus will be given to Oracle Streams, as it is one of the main technologies used in the LCG3D project.

This thesis's analysis will help experiment and grid software managers to decide what type of distribution technique should be used in their projects and estimate hardware, network, and configuration needs for a given performance expectation. This study will stand as a benchmark analysis for any FroNtier setup.

A broad study of the state of the art should be done for the scientific areas involved in this work: grid computing, database distribution techniques, and performance analysis.

1.4 Structure

Chapter 2 briefly introduces grid computing and presents currently available solutions for grid implementations with main focus on the solution used in the LCG.

Chapter 3 describes the database distribution techniques that were studied by the LCG3D project.

Chapter 4 presents the methodology used in this work including an overview on Linux monitoring tools and the description of the performance test framework developed for this study.

Chapter 5 describes in detail the experiments executed for the analysis, the respective results obtained, and the analysis conclusions.

Chapter 6 summarizes the analysis' conclusions, presents this thesis' major contributions and some directions for future work.

Chapter 2 - Grid computing

Grid computing is a high throughput computing model based on the integration, virtualization and management of distributed resources. A typical analogy is made with the electrical power grid that offers the same characteristics as grid computing: unlimited ubiquitous distributed power, transparent access, easy to plug in, a hidden complexity of the infrastructure, etc. In a grid, the user does not need to define where the data resides or what computers execute the jobs. This is again analogous to the electric power grid where the user does not know where the generator is, nor how the electric grid is wired. Grid computing is distinguished from conventional distributed computing by its focus on large-scale resource sharing.

Grid computing is used in many different application domains that usually demand heavy computation resources like simulations, forecasts, or analysis of huge amounts of data. Some typical examples of grid applications domains are particle physics, aeronautics, bioinformatics, biomedicine, weather forecast, business market simulation, etc.

Grids are either data or computing centric depending on application needs. Data grids and computing grids are terms commonly used to designate this difference. From a different perspective, grids' focus is not performance but rather throughput, i.e., the objective here is to maximize the number of executed jobs rather than optimize the speed of single jobs.

According to Foster, Kesselman and Tucke [6], grid technologies and infrastructures support the coordinated sharing of diverse resources among distributed, heterogeneous, and dynamic Virtual Organizations (VO). VOs are a key concept in grid computing. A VO is an abstract entity that enables heterogeneous groups of individuals to share resources in a controlled way, so that its members can collaborate in achieving a shared goal. Typically, these groups of individuals are not part of the same administrative domain. A Resource Provider is a facility offering resources (CPU, storage, or network) to other parties (VOs) according to specific agreements. These agreements between Resource Providers and VOs are the foundation of any grid infrastructure. They control the sharing of the resources, defining what resources are shared, who is allowed to use them, and the conditions under which sharing occurs.

From a complementary point of view [7], a grid can be seen as the integration of services offered to users across distributed, heterogeneous, dynamic VOs. Proving this is the fact that grid architectures are commonly service oriented architectures where services are offered by different components. A service is a network enabled entity that provides some kind of functionality used directly by VO users or by other services. Grid architectures consist of definitions of protocols and services.

Interoperability and standardization are key features in grid technologies. Grid infrastructures rely on very heterogeneous systems with resources provided by different organizations. Interoperable solutions built on standards enable grid components to be seen as a large single virtual computing system offering a variety of virtual resources. Interoperability is achieved via the use of protocols that define basic resource sharing mechanisms.

Grid infrastructures rely on one or more grid middleware software packages that offer a set of services such as job submission, security, resource management, accounting, data

management, information services, etc. In a lower level, grids rely on resources that are specific to fabrics like transport protocols, name servers, batch schedulers, security infrastructures, site accounting, directory services, etc.

The user interface with a grid is made through the grid middleware. The middleware allows the user to submit his job, read the status of the job, and get the output when finished. Typical steps following the submission of a job are job allocation (where will the job run), data access (where will the input data be), authentication, job execution, job monitoring, problem recovery, etc.

2.1 Grid projects and middleware

The LCG architecture is based on a set of services and applications running on a grid infrastructure [8]. This infrastructure is provided by the Enabling Grids for E-sciencE project (EGEE) [9] in Europe, the Open Science Grid (OSG) [10] in the US, and the Nordic Data Grid Facility [11] in the Nordic countries. Although these projects share many commonalities, each of them offers a different grid environment with specific grid operations, management boards, and most important, specific base set of middleware tools.

Most of the LCG computing sites and respective resources are part of the EGEE grid. EGEE is a consortium of national and regional grid infrastructures and computing centers. These entities together offer a single unified grid infrastructure with the same base middleware, and a cooperative grid operation.

EGEE was designed to extend the grid set up by the LCG project. Starting with a working grid for particle physics, the EGGE project expanded it to support other sciences by deploying a robust grid infrastructure for science. Presently, it interconnects almost 200 sites in almost 50 countries around the world, integrating several national and regional Grid initiatives in Europe, such as, INFN Grid in Italy, DutchGrid in the Netherlands, and GridPP in the UK. Currently, EGEE includes around 25000 CPUs and a storage capacity of 5 PB.

There are two middleware software packages currently deployed on the EGEE infrastructure: LCG-2 [12] and gLite [13][14][15]. Both are suites of functional components that provide a basic set of grid services. LCG-2 is the former middleware implementation released by the LCG project: it is still widely deployed on EGGE. gLite, the successor of LCG-2, is developed inside the EGEE project. It is designed to be lightweight and very modular (it has 224 modules). These two software packages co-exist in different sites of the LCG infrastructure.

These grid middleware implementations (gLite and LCG2) make use of external grid software toolkits. The Globus Alliance [16] is a community of organizations and individuals developing fundamental technologies for grid computing. One of the main projects of Globus Alliance is the Globus toolkit [17]. The Globus toolkit is an open source software toolkit that implements a basic set of grid protocols and services used for building grid systems and applications. The Virtual Data Toolkit (VDT) [18] is a grid middleware package that provides some components to both LCG-2 and gLite middleware. Conversely, some LCG-2 and gLite components are also part of VDT.

2.2 Grid services

As seen before, a grid infrastructure provides a set of grid services implemented by the grid middleware. This section will briefly describe the typical grid services offered by most middleware toolkits, and the implementations available in the LCG.

There are numerous grid middleware products capable of providing some of the fundamental grid services, as job submission and management, grid data management, and grid information services. The problem of interoperability has grown with the deployment of different middleware packages and the inexistence of widely accepted, implemented and usable standards.

The Open Grid Services Architecture (OGSA) [19] is a service-oriented architecture for grid computing. This architecture tries to define a set of core "interfaces, behaviors, resource models, and bindings" that can work as a standard for grid computing. The OSGA defines the grid as a set of services and defines the behaviors those services should have. OGSA was developed by the Open Grid Forum [20] community that leads the standardization effort of grid computing technologies. OSGA defines models and standards for grid computing widely based on web services technologies like WSDL and SOAP.

The Open Grid Services Infrastructure (OGSI) [21] proposes an infrastructure for implementing grid services where, more interesting, the Web Services Resource Framework (WSRF) is defined as one key innovation with possible impact in other fields: the stateful web services. Web Services Resource Framework (WSRF) defines a stateful web services infrastructure to model, access, and manage the state of a web service, to group services, and to express service faults.

The next sections will describe OSGA services and how is this architecture implemented in LCG by LCG2 and gLite middleware packages.

2.2.1 Job Management Services

Job management services are basically responsible for job submission and workflow control in the grid. These services address problems related to job execution, including their placement in appropriate computing facilities, submission, and lifetime management. Usually these tasks include:

- Find candidate locations for execution - check what are the locations where a job can execute; check resource requirements and restrictions such as memory, CPU, available libraries, software dependencies, and licenses;
- Selecting the execution location - match submission requests and available resources by applying different selection algorithms. These algorithms can enforce different policies or service level agreements, and optimize different objective functions like time, cost, reliability, resource availability, etc.;
- Preparing for execution – setup the execution environment includes deployment and configuration of binaries and libraries, copy input data to the execution location, etc;
- Start the execution;
- Managing and monitor the execution – includes monitoring the state of the execution, implement the appropriate recovery measures in case of failure, etc;

2.2.1.1 Workload Management Service

At a top level, these tasks are performed by a Workload Management System (WMS) that encapsulates all aspects of executing a job in a Grid environment. A WMS is responsible for accepting and satisfying job management requests coming from its users. The WMS uses other services to accomplish its task, like the execution planning services used to select the execution location of a job.

The user interface with the WMS is done using one of the available variations of the standard Job Submission Description Language (JSDL) [22]. The JSDL (commonly called JDL) is one of the most important artifacts of these services. It is a document submitted by the user to the WMS that includes: the job description, the location of the binaries, the requirements for the execution environment, the input data files or values, the dependencies, etc. A JDL document is a record-like structure composed of a list of attribute-expression pairs.

The implementations of the WMS available in the LCG are the LCG2 Resource Broker and the gLite Workload Manager Service [23]. Both provide the facilities to manage jobs (submit, cancel, suspend/resume, signal) and to inquire about their status.

The WMS is one of the key components of the middleware as it interacts with several other grid services in different ways. For example, configuration services are used to prepare the job execution environment; the information services (the databases with metadata about resources) are used to get all types of information about execution locations, policies, etc.; and bookkeeping services are used to maintain the status of the job.

2.2.1.2 Computing Resource Services

In terms of computing power, the end points of a grid infrastructure are compute farms. These farms can be computer centers with thousands of processors, or small batch systems with a few processors. These computing infrastructures are managed by batch systems, or Local Resource Management Systems (LRMS), that distribute and load-balance the CPU resources in a given farm.

LRMS have different setups and configurations. To deal with this heterogeneity, a set of services lie between the grid and the LRMS, working as grid interfaces to the LRMS. This interface is called Computing Element (CE). A CE provides services for:

- job submission to the respective LRMS – it receives a computation request from a WMS and sends the job to the respective computing resource in the local farm;
- publication of information about the CE through the grid information service – grid information services are then used by WMS to select execution locations;
- publication of accounting information in the accounting services;
- job monitoring and trace mechanism by which job status can be obtained – update log and bookkeeping services so that WMS can query the state of the jobs being executed;
- authentication and authorization mechanisms (see section 2.2.4.1).

In terms of submission there are two basic ways the CE can work:

- push model, where a job is pushed to a CE for its execution;
- pull model, where a CE asks a known WMS or a set of WMS for jobs.

A CE can be used directly by the end user or through the WMS that submits a job to a selected CE.

Both LCG2 and gLite offer implementations of a CE that handle typical job management functionalities. Both implementations rely on the Globus Gatekeeper [17] to support interfaces with several LRMS like BQS, Condor, LSF, etc. The LCG2 CE uses the Globus Resource Allocation Manager for submitting jobs to the LRMS, while the gLite CE makes use of Condor-C. Both LCG-2 and gLite CEs interface to the logging and bookkeeping services (see section 2.2.3) to keep track of the jobs during their lifetime.

2.2.2 Data Management Services

Grid jobs need to access and move data stored somewhere on the grid. Data Management Services offer the users ways to store, access, and transfer data on the grid.

Due to the typical fragmentation of a grid system, and to improve availability and performance, data is commonly replicated in multiple locations across the grid. In order to manage data replication, the grid provides file catalogues that contain metadata about the file, like replica location.

Data Management Services can be divided in three categories: storage management, data distribution, and file and replica management. Storage management includes the management of the storage systems and the access to data. Data distribution services offer data transport and placement functionalities. File and replica management services implement metadata catalogues (file and replica catalogues) used to retrieve information about data resources on the grid.

Data services should operate with generic data and do not assume any specify data semantics. Thought data services may refer to generic file types, like flat files, databases, streams, etc, in LCG these services are usually regarded as managing flat files. For example, for database systems, specific services are setup up (see section 2.4).

The following sections describe in more detail each of the three types of data services.

2.2.2.1 Storage Management Services

Storage management services control the provision of storage to applications and other services in the grid by implementing the interface with the storage resources. This interface allows grid applications to access the heterogeneous storage facilities available on the grid in a standard way. The service that encapsulates the storage system and the interface to it is called Storage Element (SE). Like computing resources (CEs), data storage resources (SEs) are the basic building blocks of a distributed computing infrastructure.

A SE is a grid service responsible for providing read and write access to data storage systems. It works as a virtualization layer for storage systems that provides local or remote space, raw or file system space, and manages quotas, file lifetime, and file properties like encryption or persistency.

The basic functionality of a SE is to provide storage space for files. This is typically achieved with a Mass Storage System (MSS), either disk space or disk space backed up by a tape system. Typically, a site provides different quality SEs, for example, a SE for transient data and another for permanent.

A SE has three external interfaces: the management interface (SRM), the POSIX-like I/O interface (Grid I/O), and the file transfer interface (GridFTP).

The Storage Resource Manager (SRM) works as an interface between the SE and the underlying storage system. An SRM interface must be implemented for each specific storage system. This allows the SE to offer a standard interface to all types of storage systems. This

SRM interface provides a common set of storage systems' functionalities (independent of the implementation) that include mechanisms such as quotas, file lifetime management, pinning, space reservation, etc.

The POSIX-like I/O, also called Grid I/O, provides users with mechanisms to hide the complexity of file I/O with a storage resource. Using these services, a user application only needs to use the Grid I/O API to directly access data on SEs. Grid I/O makes the interface with the SE, the SRM and the security services on behalf of the user to get POSIX-like I/O access to files on the SE. Typically, these tools also include interfaces with grid file catalogues to enable an application to open files not based on their real name in the SE but rather based on logical file names (see section 2.2.2.3 for more details).

The file transfer interface is supported by a file transfer protocol like GridFTP to enable data transfer between the grid and the SEs. The GridFTP [24] protocol is an extension of the FTP protocol designed for grid environments. It includes features such as being compliant with the Grid Security Infrastructure (GSI) (see section 2.2.4). Normally this transfer service is invoked indirectly through the File Transfer Service (see section 2.2.2.2) or through the use of the SRM interface.

Additionally, a SE has to provide authentication, authorization, and accounting/ auditing facilities. Data stored in a SE must be protected by Access Control Lists (ACLs). This is implemented by the SE using the grid security services described in section 2.2.4. On the other hand, the SE must provide sufficient information for tracing and auditing user activities. The SE is also supposed to produce monitoring information about the usage of the storage system according to a predefined schema.

Both LCG2 and gLite implement a SE. These SEs consist of SRM, a GridFTP server for file transfer, and a tool for POSIX-like access to the data.

The LCG2 Disk Pool Manager (DPM) [25] is a lightweight implementation of a SRM interface to disk only storage systems. gLite itself does not provide a SRM; instead, external SRM implementations are used. The CERN Advanced Storage Manager (CASTOR) [26] is a storage manager that, unlike DPM, provides a SRM interface to both disk and Mass Storage Systems (MSS). CASTOR emulates a distributed file system and an associated tape storage system. It is the most commonly used SRM in LCG, though DPM is used for smaller and disk only grid sites. Other mass storage management systems for tape/disk systems in use in LCG include dCache [27], the High Performance Storage System (HPSS) [28] and IBM Tivoli [29].

LCG2 implements its own GridFTP interface to disk storage. Like the SRM, gLite itself does not provide a GridFTP server; standard implementations like GridFTP server of the Globus toolkit are used instead.

LCG2 and gLite themselves do not provide mechanisms for direct access to data on a SE; instead, external packages like RFIO or dCap are used. In LCG2, the Grid File Access Library (GFAL) offers a POSIX-like I/O abstraction layer over these packages for accessing files in a SE. It provides access to files via their logical name by interfacing with a file catalogue, and enforces access control lists specified in the catalogue if appropriate. In gLite, these functionalities are implemented by the gLite I/O service [30]. gLite I/O currently interfaces with the FiReMan and the LCG-RLS catalogues (see section 2.2.2.3 for details on these file catalogues).

2.2.2.2 Data Distribution Services

Data Distribution Services are data transfer and replication services. These services provide data transfer mechanisms between resources, and a transfer control interface by using appropriate protocols. A transfer can create a copy of the original file (replication) or migrate it completely. Data distribution services are supposed to provide scalable and robust managed data transfer between grid sites, to and from SEs. They should provide ways to specify the Quality of Service (QoS), such as reliability of the transfer, the maximum bandwidth to use, the required delivery time, or delivery guarantees.

Typical use cases of data transfer services include: movement of big chunks of data between sites, for example, in LCG the distribution of data from Tier-0 to other tiers; movement of files between SEs to push data closer to the CE where a specific job is running; and staging of copies of files to the local machine where the job is running (done by the WMS to avoid remote access to data on the SE).

File placement and replication services are services that manage the location of multiple copies of data either to increase availability through redundancy or to improve performance by reducing access times. These services provide a layer above the basic file transfer service and implement routing and replication policies. They are quite rarely implemented as these policies heavily depend on the application. For this reason, the data transfer services are usually used directly by client applications.

LCG2 does not provide a File Transfer Service. The user is responsible to issue the relevant commands to replicate files between SEs, for example, using GridFTP that provides basic-level data transfer services between grid sites.

Though, in LCG, file placement and replication services are seen as an experiment responsibility, the gLite File Transfer Service (FTS) [31] accepts data transfer requests and executes them according to defined policies. Additionally, the File Placement Service (FPS) makes use of file catalogues (see section 2.2.2.3) to provide logical file naming functionalities.

2.2.2.3 File and Replica Management Services

File and Replica Management Services manage the description of data available in the grid and provide lookup services in form of catalogues. Basically, these file and replica catalogues consist of a list of files, replicas, and respective locations in the grid.

A File Catalogue Services usually includes the following features:

- File naming resolution – conversion from Logical File Names (LFN) to physical storage locations;
- Hierarchical namespace – storage of directory structures;
- Access control to the catalogue – directory level access control that respects the Access Control Lists (ACL) specified by either the user creating the entry or by the catalogue administrator.

With this set of functionalities, grid users or applications do not need to know where the files actually are, and, instead, use LFNs to refer to them. The file catalogue can then be used to translate the LFN to a physical location.

Additionally to these functionalities, a typical use case for the file catalogues is the request by the WMS for locations of SEs that contain a file (specified by a LFN). This info is used by the WMS to determine which sites contain the data that the job needs.

Metadata catalogues are data services that store descriptions of data held in certain other data services. Though the grid Information Services (see section 2.2.3) are responsible for holding metadata about the available services on the grid, application specific metadata can be stored in metadata catalogues. Metadata catalogues are typically implemented separately from file and replica catalogues except in catalogues for file-based metadata, where the metadata catalogue is seen as an extension of the file catalogue.

Both in LCG2 and gLite, a set of command line tools called lcg-utils can be used for operations with the catalogues and files such as catalogue querying and file replication.

LCG inherited the Replica Management Service (RMS) from the European Data Grid (EDG) project. The RMS provides a single interface for replica location and metadata catalogues that maintain information about the existing replicas of a file and the mappings between file identifiers and LFNs. This service is being phased out in LCG and, instead, LFC (see below) is being used.

The basic functionalities of the LCG File Catalogue (LFC) are the direct translation of logical file names to physical file addresses, and the identification of the site where a given file resides. It offers a hierarchical view of the logical file name space and provides UNIX style permissions and POSIX Access Control Lists (ACL). LFC supports metadata services by associating key/value pairs with file entries. The gLite File and Replica Catalogue (FiReMan) [32] offers the same set of functionalities as the LFC, together with some additional features like the support for bulk operations. Recently, LFC has been chosen as the supported file catalogue in LCG.

2.2.3 Information, Monitoring, and Accounting Services

Information services are a vital low-level component of a grid as most of the services in the grid will use information services, either to publish information, or to consume information. Information services implement mechanisms for accessing and manipulating information about applications, resources and services in a grid environment. The term information refers to data used for discovery, status monitoring, accounting and logging. These types of data map to the four basic capabilities of an information service: monitoring, accounting, discovery, and logging.

The monitoring services gather information from the different sites about all types of grid services, and publish them. Published information can be from service usage statistics to resource status information. These services are used by user applications, resources, and services themselves.

The accounting services keep information about the usage of resources by the users by tracing single transactions, like jobs or file requests. These services are used to produce statistical reports, track resource usage for individual users, and to discover abuses. Accounting information can be used, for example, to charge users for the resources they use, and to implement resource sharing policies based on user quotas. Monitoring services use accounting information, but the information and functionalities provided by monitoring systems go beyond accounting information.

Service and resource discovery is a service offered to end users and to other services to locate services available in the grid. Though it is designed primarily for queries, a mechanism for service registration has to be associated. The service discovery service is typically a front end to a directory system as it needs to be optimized for searches. Directory systems are easy

to replicate, offer bw latency response to high volumes of queries, and typically have an associated caching mechanism.

The logging and bookkeeping services track jobs during their lifetime using information gathered from the CEs. The logging service stores log records in a persistent database for a long period of time. Logging data includes job status history, job execution statistics like CPU consumption, memory usage, etc. The bookkeeping service stores actual information about running jobs. It stores richer but more volatile data than the logged data. It acts as an intermediary between log producers and consumers. Bookkeeping data includes important points of the job life-cycle, the job description, the input and output file names, etc.

An example of use of information services in a grid: a data management service such as FTS (see 2.2.2.2) publishes raw information about a file transfer in the monitoring information service; this service is responsible for providing monitoring information such as bandwidth, utilization patterns and packet size based on information published by other services; this info is used by the WMS (see 2.2.1.1) to decide how to move a given file, respecting the agreed QoS.

Though it is possible to use a single information service covering all available information about a grid, typically, different information services are implemented separately, for example, the information services for accounting are implemented with logging and bookkeeping servers, while the service discovery service are implemented with a directory system.

An information service has an associated data model and query language. Different grids use different data models and query language for the information services. The most common solutions are based on XML and XPath/XQuery query languages, or on relational models and the SQL query language.

Information services publish and maintain data about resources in grids. In LCG, this information is modeled after the Grid Laboratory Uniform Environment schema (GLUE) [33]. The GLUE schema is an abstract model of grid resources that describes the resources available at a site and the current state of those resources. Grid Information Services use concrete schemas based on the GLUE schema.

In LCG, monitoring services are implemented with the Relational Grid Monitoring Architecture (R-GMA) information service [34]. R-GMA is an implementation of the Grid Monitoring Architecture of the Open Grid Forum [20]. It presents a relational view of the collected data. It is basically a producer/consumer service: it models the information infrastructure of a Grid as a set of consumers (that request information), producers (that provide information), and a central registry which mediates the communication between producers and consumers. R-GMA makes all the information appear like one large relational database that may be queried to find the information required. R-GMA is currently also used to collect LCG accounting records (see below). Both LCG2 and gLite use R-GMA.

In LCG, service discovery is implemented either using R-GMA or the Berkeley Database Information Index (BDII). BDII is a scalable grid index information service which collects discovery information from grid resources. It consists of a set of standard LDAP databases. BDII follows the GLUE information model. Both BDII and R-GMA are used for monitoring with LCG2: information providers inspect the status of grid services and publish their data into BDII or RGMA. The gLite Service Discovery API [35] provides implementations of service discovery interfaces to access the R-GMA and the BDII information services.

LCG2 accounting data is collected by the Accounting Processor for Event Logs (APEL) system [36] which publishes its data into the R-GMA system. The gLite Accounting System (DGAS) [37] is a full featured grid accounting toolkit that collects information about usage of grid resources. This information can be used to generate reports but also to implement resources quotas. DGAS integrates with the APEL system by feeding the R-GMA repository using the APEL data format.

The gLite Logging and Bookkeeping services (LB) [38] tracks jobs managed by the WMS. It gathers events from various WMS components and processes them in order to give a higher level view of the status of job (with execution conditions and environment). Additionally, gLite offers the Job Provenance (JP) service [39] that provides long-term permanent storage of job related information (as stored in the LB) and allows end users to use this data for data-mining, debugging, post-mortem analysis, etc.

2.2.4 Security services

Grid security services functionalities include:

- identification of users, systems, and services (authentication);
- access control to services and resources (authorization);
- information provision for analysis of events related to security (auditing);
- implementation of security-related VO policies;
- data transfers' encryption and integrity checks;
- delegation mechanisms;
- mapping of global grid identities and roles to resource specific identities and roles.

The Grid Security Infrastructure (GSI) [40] of the Globus Toolkit (see section 2.1) implements a collection of grid security services that follow the OGSA architecture (see section 2.2). Although many models for grid security exist, the most used, and the one used in GSI, is based on public key cryptography (or asymmetric cryptography). GSI provides libraries and tools for authentication, authorization, delegation, and message protection that use the Public Key Infrastructure (PKI) based on standard X509 certificates, the SSL/TLS protocol, and X509 proxy certificates (an extension to X509 certificates to accomplish delegation requirements of grid communities).

With the PKI encryption schema [41], a user has a private key that he uses to encrypt messages. These messages can be decrypted by anyone using his public key. Using this method, a user can "digitally sign" his messages, i.e., assure the receiver that the message was not altered since it left the sender.

Another essential concept connected with the PKI is that of a certificate. A certificate identifies every user and service on the grid, and contains information to authenticate the user or service. A certificate is issued by a trusted third party called a Certificate Authority (CA). The role of the CA is to issue (digitally sign) user certificates. A CA is usually run by a large organization or commercial company because it must be trusted both by the sender and by the receiver of any communication. A certificate is an authentication credential where the CA asserts that a given name and a public key belong to the referred subject. Thus, a certificate includes the subject name, the subject's public key, the identification of the CA that issued the certificate, and the digital signature of the CA. The standard X509 certificate format is the most widely used for this purpose.

In a grid where the standard X509 certificates and the PKI are used, each user holds a unique authentication credential, i.e., a X509 certificate issued by a CA. This credential is the basis for all the security functionalities available in the grid.

2.2.4.1 Authentication and Authorization

Authentication is concerned with identifying users, systems and services by verifying some kind of given proof of identity. It is used whenever a context for message exchange needs to be established in a grid. Examples of authentication procedures include: the evaluation of a username/password pair; the authentication through a Kerberos mechanism where a ticket is passed to the service provider that determines the authenticity of the ticket; and, the most common in grid environments, the use of the X509 certificates.

In LCG, authentication functionalities are based on GSI, i.e., on PKI and X509 certificates. The CAs that issue the certificates in LCG are accredited by entities called Authentication Policy Management Authorities. Users and services can request a certificate to one of the accredited CAs. This certificate is then used for authentication in all resources in the grid.

Authorization services are responsible for implementing access control policies. Given an authenticated user and a specific resource that the user wants to access, the authorization service decides whether or not the user is authorized to access the resource. A user may be member of any number of VOs, and a VO can have a complex structure with groups and subgroups. A VO gives its members access to various types of resources and grid services are responsible for respecting this access policies. To accomplish this, the system must keep information regarding the relationship of the user with the VO (groups and roles), and what is the user allowed to do at specific resources owing to his membership of a particular VO. In a grid environment, it is not possible to administer authorization information on a site basis as users have administrative deals with the organization they work in and with their local site only. To solve this problem, two kinds of authorization services exist: policy assertion services and attribute authorities.

Policy assertion services give a user (or a set of users) the explicit privileges to perform an action (or set of actions) on a specific resource (or set of resources). These services are implemented by policy statement services like the Community Authorization Service [42].

Attribute authorities services issue attribute certificates that associate a user with a set of attributes (roles, group membership, etc.) in a trusted manner. The resource uses these user attributes, asserted by the attribute authority, when evaluating user access requests. Public key certificates are meant for authentication while attribute certificates for authorization. Although there is a standard defined for X509 Attribute Certificates [43], instead of a separate attribute certificate, an extension of the public key certificate can be used to place the authorization attributes. Both solutions offer advantages and disadvantages. In LCG, authorization services implement access control functionalities based on extended X509 certificates that contain user attributes based on roles and groups taken from the VO Membership Service.

The Virtual Organizations Membership Service (VOMS) [44] is the most popular system for managing authorization information in grid infrastructures. It is used to manage the information about user's roles, and user's membership to groups and sub-groups inside a VO. In a first step, the VO manager assigns the user these roles and memberships; with this information, VOMS provides a service to generate extended user certificates containing the

credentials stored in the VOMS server, i.e., VOMS is an attribute authorities service. Although X509 Attribute Certificates could be used, VOMS attributes are embedded in the users' certificate, enabling the client to provide authentication and authorization credentials in a single operation.

VOMS is implemented with a client-server architecture where the VOMS server is basically a front-end to a RDBMS with all the information about user accounts and VOs. The VOMS client allows users and administrators to access and manage the server repository. Administrators can also manage VO membership with the VOMS administrator web interface. The VOMS client can also be used to generate user certificates. These certificates are issued using information from the VOMS server, and include: user credentials, VOMS server credentials, time of validity, and authorization information. The authorization information contained in the certificate extension can be read by grid services (WMS, CE, file catalogues, etc.) to give or not the user access to its resources. It is commonly used to maintain ACLs.

Both LCG2 and gLite use VOMS for authorization information management. Access control to resources is implemented with VOMS authorization attributes stored in the certificate of the user.

2.2.4.2 Identity and credential mapping

An identity mapping service is responsible for transforming an identity that exists in one domain into a significant identity within another domain. This service is concerned with user name mapping issues rather than user authentication/authorization. It is used when a resource do not understand grid credentials natively, but instead use a different mechanism to identify users. For example, a user may have an X509 certificate for authentication against a grid CE; the identity mapping service would map this certificate to a site specific identity that has meaning inside that particular CE environment, for example, an UNIX account.

Credential mapping services implement the same functionality but applied to credentials, i.e., group membership, privileges, and roles associated with a given entity. For example, the credential conversion service may convert an attribute certificate into a form of attribute list that is understood by the authorization service. This functionality makes the different credential types interoperable. Additionally, these services implement credential conversion policies, and depend on the identity mapping service.

VOMS (see above) provides mechanisms for identity mapping by using user roles. A user certificate issued by VOMS includes authorization information that is used, for example, by the CE to map the grid user identity to a local user and group.

In addition to VOMS, local site authorization grid services like LCAS and LCMAPS can be used [45]. The Local Center Authorization System (LCAS) is a service used locally on sites to enforce local security policies. The Local Credential Mapping Service (LCMAPS) is a framework that can load and run credential mapping plug-ins. LCMAPS can map users to local accounts (like UNIX, AFS, or Kerberos). Both LCAS and LCMAPS interoperate with VOMS servers. The Grid User Management System (GUMS) [46] is a grid identity mapping service that performs identity mappings and communicates them to the CEs. GUMS is used in the Open Science Grid (OSG), and it is used by gLite via a dedicated LCMAPS plug-in.

2.2.4.3 Proxy certificates

User single sign-on is one of the basic objectives of a grid authentication service. Grid authentication services should provide an identity credential with universal value in the grid that works with the many different systems, services, etc.

Delegation is used by grid users to temporarily grant a subset of their privileges to another entity. Examples of delegation include: a user that needs to move a file needs to grant file access rights to a file transfer service, so that the service can perform the file transfer on his behalf; a user grants privileges to a process that must run without intervention; brokering services that acquire resources on behalf of the user; etc.

Standard X509 public key certificates do not include capabilities to implement these features. The most widely adopted mechanism in the grid community, including the GSI, to implement single sign-on and delegation capabilities is the use of X509 Proxy Certificates (PC) [47].

X509 Proxy Certificates allow any user to dynamically assign a new identity to an entity, and to delegate a subset of his privileges to that identity. A PC is an X509 certificate signed, not by the CA, but by the user himself. A PC has a unique identity derived from the identity of the user, and a distinct public and private key pair. It may contain authorization information (controlled by the user or by some auxiliary service) that is placed in one of the X509 extensions available. The schema is recursive as a PC can be used to sign a new PC.

PCs can be used for delegation: the user generates and sends a PC to a given service; whenever the invoked service takes actions on behalf of the user, the service uses the PC to delegate the privileges included in the PC. Services that act on behalf of the user shall use PCs derived from the original user certificate.

PCs can be used for single sign-on: the user, after creating a PC, keeps a new certificate and a private key. This private key does not need to be encrypted as the proxy has a short lifetime: file system level protection is typically enough. In this scenario, the user, in order to create a PC, has to type his password to access his long term private key. After this, the non-encrypted private key of the PC is used to authenticate in all services needed.

To authenticate in a service using PCs, the user sends his X509 certificate and a PC to the service. In this scenario, two verifications are done by the service: the user's public key (taken from the user's certificate) validates the signature on the proxy certificate, and the public key of the CA validates the signature on the certificate of the user. This mechanism establishes a chain of trust from the CA to the proxy through the user.

In LCG, VOMS use these authentication and delegation mechanisms provided by GSI. The proxy certificates used in LCG are, in fact, VOMS certificates that are issued by a VOMS server. These proxy certificates have a special format that includes an authorization extension. A user, before starting a job in LCG, must acquire the VOMS proxy certificate that is signed by his VO and only valid for a limited period of time. The extra authorization information is placed as a non-critical extension in the proxy, so that these certificates can be used with services not aware of VOMS.

MyProxy [48] is a credential repository that helps the user managing and protecting files containing sensitive data like private keys. MyProxy is a rather simple service that stores user X509 PCs for later access. Additionally, the PCs are protected by a pass-phrase, and users and administrators can set access control policies on the credentials in the repository.

2.2.4.4 Audit and secure logging

The audit services are responsible for producing log records of security-related events. These audit records are analyzed by VO security administrators to determine whether defined security policies are being followed.

Having shared requirements with the accounting services, auditing services enable VOs to determine precisely who did what, where and when. The audit process consists of enforcing uniformity of audit information, easing the process of combining audit information from different sources, and identifying abnormal events registered in the logs.

Although a dedicated audit service would bring some benefits, typically, auditing does not require a separate grid service. Instead, logging and accounting services are used for imposing a set of logging policies. In LCG, the gLite Logging and Bookkeeping services (see section 2.2.3) are used for auditing purposes.

2.2.5 Conclusion

Though, in this section, we only mentioned the LCG-2 and the gLite middleware implementations, several other implementations co-exist in EGGE, OSG, NDGF, and other grid projects around the world. Interoperability between middleware implementations is an essential component of the system. Grid services must be provided in a way that applications should not need to be aware of which grid infrastructure they are running on. EGEE, OSG and the Nordic Data Grid Facility share a lot of commonalities in terms of grid services implementations (like CEs, SEs, WMSs, information system schemas, etc) but they are still not completely interoperable. There is a strong collaboration between the institutions involved to ensure that these grid infrastructures can interoperate transparently.

Grid middleware packages must have a well defined life-cycle support. This process manages the integration of the middleware components into a coherent distribution. It includes the testing and certification of middleware components, the management of the deployment process (update procedures, security fixes, and configuration management), the establishment of feedback loops, and general maintenance.

Application domain specific considerations can be taken into account when designing grid middleware components. Typically, grid infrastructures offer additional functionality to fulfill the needs of different application domains. In the LCG, the ARDA project investigates the area of distributed analysis together with the LHC experiments, and is meant to influence the evolution of the LCG middleware packages in that respect.

2.3 LCG infrastructure

Although grid services are conceptually nearly the same in all modern grid infrastructures, there is a big number of specific characteristics on every grid infrastructure. This section describes the LCG infrastructure as an example of a grid infrastructure.

As seen in section 1.1.4, the LCG is implemented in a distributed tiered model. The Tier-0 at CERN, will held: the data acquisition, part of the data processing, and the data distribution to Tier-1 centers, where most of the data analysis will be performed. Tier-2 and Tier-N sites will held simulation processing and end-user data analysis. All these sites together will form the LCG infrastructure where grid users will profit from the available grid services to analyze the data acquired in the LHC experiments.

All sites in LCG will provide basic grid functionality being that Tier-1 centers will provide a more complete set of grid services. Each Tier-1 centre will have specific agreed obligations to support particular VOs. Additionally, Tier-1s must provide required services, like archival storage resources, to Tier-2 centers.

Typically due to specific design or development options, in a grid, not all VOs (the LCG experiments in the case of the LCG) rely on the complete set of grid services. It is common to have in a grid infrastructure, not only different middleware implementations, but also VO specific grid services that interoperate with the standard services and tools provided by grid infrastructure.

Production-like exercises are performed periodically in the LCG infrastructure to exercise its capacity and limits [4]. In December 2005 there was a data recording test at CERN that moved data from a simulated generator to disk, and then to tape. The data rate of 750 MB/s was sustained for one week. Similarly, in April 2006, a data transfer test from Tier-0 disk storage to Tier-1 tape storage sustained a transfer rate of 1,6GB/s for three weeks. Both tests were performed using the CASTOR mass storage system.

In the LCG, since the middleware packages and the operational policies of the infrastructure projects are different, the operational control is the responsibility of each of the different grid infrastructure projects: EGEE (Europe, Asia, and Canada), Open Science Grid (USA), and Nordic Data Grid Facility (the European Nordic countries). A huge effort for the coordination of these operations is needed.

Grid support services are typically divided in user support, and grid operational support. The Grid Operations Centers [49] provide essential grid operational support services, such as maintaining configuration databases, operating the monitoring infrastructure, and controlling the accounting system. The Grid Call Centers provide a user support structure that covers both the grid and the computing service operations.

A basic requirement in the LCG infrastructure is the need for very good network connections between sites. LCG Tier-0 and Tier-1s are inter-connected by dedicated 10Gbit links. Other tiers are inter-connected by general purpose research networks.

The complete list of LCG Tier-1 sites is show on table 1.

Tier-1 name	Tier-1 location
ASCC	Taipei, Taiwan
BNL	Upton, NY, USA
CERN	Geneva, Switzerland
CNAF	Bologna, Italy
FNAL	Batavia, IL, USA
IN2P3	Lyon, France
GridKa	Karlsruhe, Germany
SARA/NIKHEF	Amsterdam, The Netherlands
NDGF	Nordic countries
PIC	Barcelona, Spain
RAL	Didcot, United Kingdom
TRIUMF	Vancouver, Canada

Table 1 - Tier-1 centers and their locations

2.3.1 LCG site architecture

Each site in LCG, independently of what tier it belongs to, has its own management boards, local policies, planning activities, budget constraints, monitoring solutions, computing and storage facilities, etc. The grid copes with this heterogeneity by offering a virtualization layer that hides these differences among sites. Nevertheless, each site is responsible for maintaining its local computing infrastructure, and implementing the low level components of the grid, for example, a batch system with which the CEs interact.

The site architecture at Tier-0 and Tier-1s is based on three functional units: computing (CPU) resources, online (disk) storage, and archival (tape) storage. The other tiers do not typically provide tape storage.

An archival storage system consists of a tape system with a front-end disk cache running a storage management system like CASTOR. An online storage system supports file level access, and so, in addition to GridFTP (also available in tape systems), a POSIX-like interface is offered. As seen in section 2.2.2.1, the services presented by storage systems follow the Storage Resource Management (SRM) interface that runs above the transport layer (GridFTP or POSIX-like interface). Storage will appear as a Storage Element on the grid. Tier-1 centers are required to archive, reprocess, and serve as necessary, a fraction of experiment's data.

Site computation services are implemented using computing farms together with a computing resource management system. An example of such a system is the batch scheduler called Load Sharing Facility (LSF) that is used in the CERN computing farm at Tier-0. LSF is a general purpose distributed batch system used to distribute and load-balance the CPU resources in a computing farm. In order to make better usage of resources available, a batch system like LSF makes a cluster of computers look like a single virtual system. With LSF, hosts from various vendors can be integrated into the same system. LSF can automatically select hosts in a heterogeneous environment based on the current load conditions and the resources requirements of the applications. LSF deals well with thousands of running and queued jobs.

Both storage and computing systems need an associated local infrastructure management system. The bcal infrastructure management system used at CERN is the Extremely Large Fabric management system (ELFms) [50]. ELFms is a low-level node management system developed at CERN that provides a consistent full-life-cycle management and high automation level. It contains three components:

- quattor [51], a powerful system administration toolkit for the automated installation, configuration, and management of clusters and farms running Linux or Solaris.
- The LHC Era Monitoring (LEMON) monitoring system [52]. On every monitored node, a monitoring agent launches and communicates with sensors which are responsible for retrieving monitoring information. The extracted samples are forwarded to a central measurement repository (see section 4.10.2.7 for more details).
- The LHC-Era Automated Fabric (LEAF) [53] toolset enables high-level commands to be issued to sets of quattor-managed nodes, and allows hardware equipment to be visualized and easily located in a computing center.

2.4 Databases and the Grid

Most middleware implementations provide an infrastructure for distributed access and replication of file based data (see section 2.2.2). Nevertheless, most of the user applications and grid services require a similar infrastructure for data stored in relational databases as many applications and services rely on Relational Database Management Systems (RDBMS). RDBMSs offer a structured, flexible, and consistent way of storing data at the cost of introducing some complexity in the data management process.

The distributed grid environment, and, in the case of the LCG, the layered grid model, imply that grid users will be distributed around many different sites in the grid. For this reason, distributed database access and the consequent need for database replication are major requirements in grid deployments. Moreover, high levels of availability and scalability are generally required for grid databases. At the bottom line, reliable database services are required at almost all grid sites whereas, small and peripheral grid sites, may rely on other bigger sites for these services.

The database distribution model for a given grid depends on the requirements of the individual applications using the grid and on the grid infrastructure itself. Different technological solutions can be used to fulfill those requirements. One of the most important application aspects, in terms of database deployment, is whether it runs centrally or locally. If the application runs centrally, replicas of those databases are typically needed across the grid. Conversely, if the application runs locally, it will probably need not only replication but also database synchronization mechanisms.

A common example of the need for database replication in a grid is the replication of file catalogues (see section 2.2.2.3) to different grid sites (e.g. in LCG from Tier-0 to Tier-1). Grid file catalogues, as most of the grid services, can run as central services in main sites, or as local catalogues in smaller sites. The deployment of mechanisms for replication and consistency maintenance is mandatory in both cases.

A LCG specific example of required database services are the experiment specific applications that use conditions databases. These databases store information about several experiment aspects that need to be indexed to space and time coordinates. Conditions data, as much of the data in LCG, will be collected at the online sites of Tier-0. This data needs to be shipped to the main off-line site of Tier-0 for backup and analysis. In order to fulfill these requirements, a database copy or replication mechanism needs to be deployed. Moreover, the databases deployment on both sites must be synchronized, i.e., RDBMS and infrastructure choices, support services, etc. must be decided in agreement between the two parties: the experiments and the LCG project.

A distributed database deployment solution implies a database support service. Database support services on a grid are different from generic grid support services and require very different and specific expertise. These support services have to cope with technological options, like the RDBMSs being used, as different RDBMSs imply different expertise.

The data volume requirements for a distributed database infrastructure are obviously essential factors on the design phase of such a system. The estimated size of the databases used in the LCG is 1% of the total file based data, which is 15 PB per year. This estimate points to 150 TB per year of database data, and an associated database backup volume of 375 TB per year. As the data volumes are extremely big and the accuracy of these estimations can be rather low, the database infrastructure must be scalable. This means that the infrastructure

must scale well, not only in terms data volumes, but also in terms of database server performance.

The setup of the LCG database distribution infrastructure and the deployment of the database services are coordinated by the project LCG Distributed Deployment of Databases (LCG3D) [5]. The goals of this project are to provide a consistent way of accessing database services at all the LCG tier sites, and to investigate distribution techniques to make this possible.

The LCG3D project offers an infrastructure for consistent distributed deployment of databases at the LCG sites that includes the deployment and test of database replication mechanisms. This infrastructure is based on an independent set of databases loosely coupled via asynchronous data replication or data copy mechanisms. More details about this infrastructure and the replication techniques used can be found in chapter Chapter 3 -. Moreover, LCG3D offers a consistent way of accessing database services in LCG through the development of the CORAL project (see chapter Chapter 3 -).

2.5 Summary

This chapter introduced some basic grid computing concepts and described a grid as a collection of interoperating services serving a group of Virtual Organizations. Concise descriptions of these services were presented along with examples of currently available implementations.

Chapter 3 - Database Distribution Techniques

Several alternatives exist to connect database clusters and replicate data in a distributed environment which provide different levels of redundancy, data consistency and scalability. This chapter will discuss the topic of database distribution techniques.

For smaller data volumes and in reliable network environments direct synchronous data copying can be perfectly acceptable (in an Oracle database [54] via database links or materialized views [55]). For larger volumes of read-only data with well-defined physical conditions, the exchange of complete tables or tablespaces as files may be an option (e.g. Oracle transportable tablespaces [56], SQLite [57] or MySQL [58] files).

The general case though is more difficult to handle as database data may be updatable, contain complex relations between different database schemas, or network problems may exist. In these cases, more advanced database distribution techniques have to be selected.

The next sections will describe the two main distribution techniques studied: Oracle Streams and FroNtier. Oracle Streams [59] is an Oracle product that can maintain consistency between relational data at different sites. FroNtier [60], combined with the web caching server Squid [61], is a database caching solution that, in cases where data is read-only, offers very good performance though some data consistency rules have to be followed.

With these technologies available, in LCG, the proposed database service architecture is summarized in Figure 1 [62] where O stands for Oracle Database, F for FroNtier server, S for Squid Cache server and M for MySQL Database.

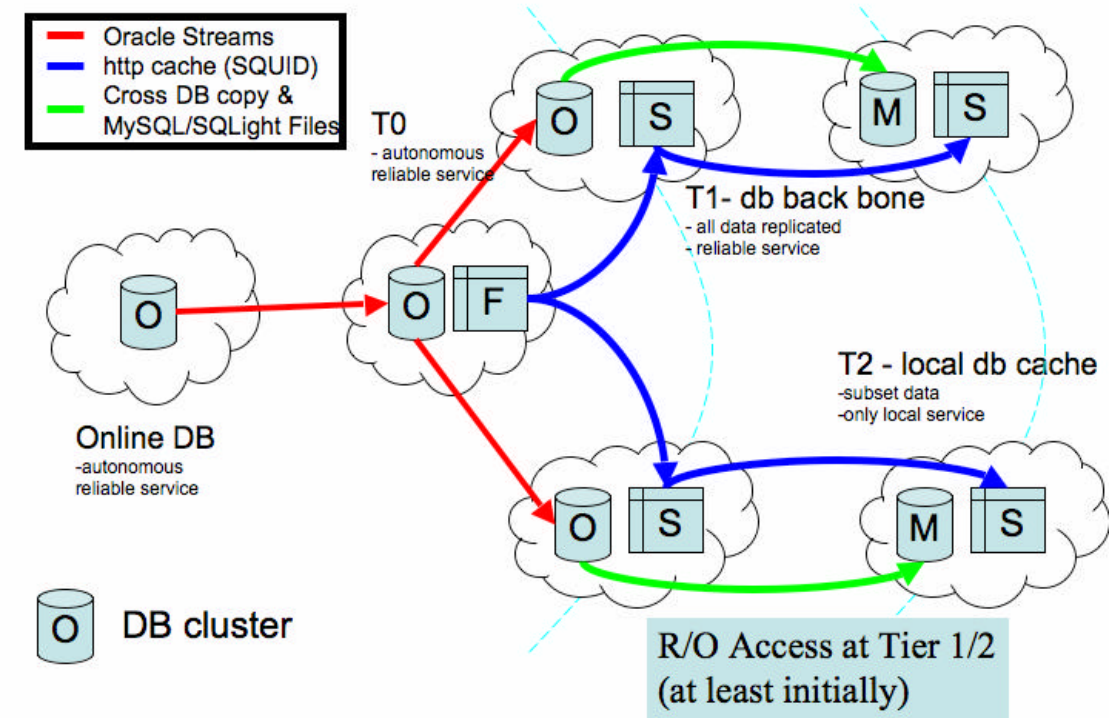


Figure 1 - LCG 3D Service Architecture

3.1 Oracle Streams

The Oracle Streams product enables the propagation of data (single tables or complete schemas) from one database to another keeping it up to date in Real Time. The data is propagated in a stream that routes the specified information to subscribed destinations.

Streams utilize the database redo logs to capture data or schema changes on the source database side (in this case, the LCG Tier-0 databases). These changes are then shipped via an asynchronous propagation process to one or more destination databases (in this case, the LCG Tier-1 databases). The changes are shipped between databases encoded inside Logical Change Records (LCRs). At the destination, these LCRs are applied in correct transactional order. Complex filter rules can be used to select what data should be propagated. In case of network problems or database service intervention, change records are kept on the source system and are automatically applied when the service is re-established.

Streams can be setup to provide a unidirectional or bi-directional connection between databases. Even though bi-directional streams have been tested successfully, they add significant complexity to the deployment as conflicts between updates on both streams endpoints may arise, and need to be handled.

A schematic description of the streams setup deployed in the LCG is shown in Figure 2.

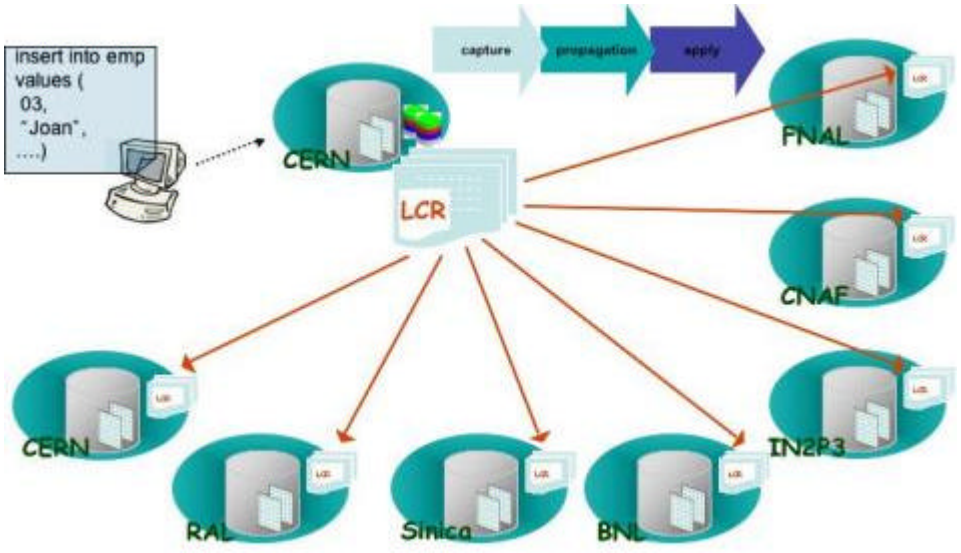


Figure 2 - Database Replication with Oracle Streams

The architecture of Oracle Streams is based on three basic components: Capture, Staging (or Propagation) and Consumption (or Apply).

Capture

Oracle Streams capture events into the staging area in two different ways: implicitly or explicitly. With implicit capture, the server does a log-based capture of DML and DDL events from the source database. After retrieving the changed data, the capture process formats it into a LCR, and places it in a staging area for further processing. The capture process can filter LCRs based on defined rules, so that only specific changes to desired objects are captured.

The database process which is capturing and queuing changes can optionally be executed on a separate machine (see Figure 3) to minimize the impact of the capture process on the source database. This is called downstream capture.

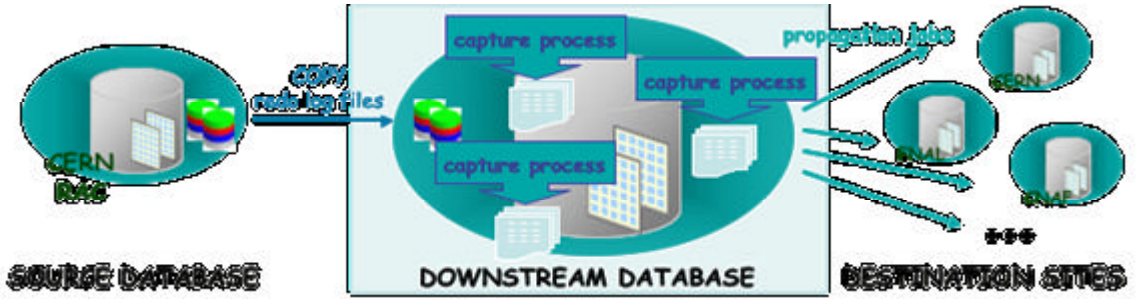


Figure 3 - Downstream Capture Setup at Tier 0

Explicit capture allows applications to explicitly generate events and place them in the staging area. User applications can explicitly enqueue user messages representing events into the staging area. These messages are formatted as LCRs, which will allow them to be consumed by the apply engine, or they can be formatted for consumption by another user application.

Staging (or Propagation)

Once captured, events are published in a staging area which is a storage queue of captured events. Subscribers, registered destination databases, examine the contents of the staging area, and determine whether or not they have an interest in an event. A subscriber can either be a user application, another staging area, or an apply process. Other staging areas can subscribe to events in another staging area, and transformations can be performed as events enter, leave, or propagate between staging areas. In the simplest case, the subscribers are apply processes from the destination databases, and events remain in a staging area until consumed by all subscribers.

Consumption (or Apply)

Events in a staging area are consumed by subscribers that are either apply processes, or applications using an API. Implicitly enqueued events can be dequeued by a default apply process, or by a user-defined apply process. The default apply process directly applies the DML or DDL represented in the LCR to a local Oracle table. A user-defined apply function can be built for specific cases.

Automatic conflict detection mechanisms between data in the source and destination databases can be setup: unresolved conflicts are placed in an exception queue. Moreover, it is also possible to define rule based configurations.

Summary

A key advantage of Oracle Streams, with respect to other mechanism such as application specific copy tools or data caching, is their simple and generic semantics. Many key LCG database applications have been validated in the streams environment, and continue to function without any application changes, or application specific replication procedures.

3.2 FroNtier

FroNtier is a data distribution technique based on the caching of database query results that can be used for read only access. Instead of providing a connected set of distributed database servers, that is hard to deploy and maintain, here a distributed hierarchy of independent cache servers is introduced between database clients and a (central) database server. The FroNtier package was developed in order to deploy a standard web proxy cache server (Squid) to cache database data.

FroNtier is a simple application that encodes the communication between database client and server into HTTP requests. It is a Java servlet [63] that runs in the servlet container engine Tomcat [64]. The HTTP requests encode select statements, and the replies encode database result sets in XML [65] format. FroNtier translates the select statement encoded in the HTTP request, executes the statement through a JDBC connection in the database server, and encodes the result set in a XML file. The operation mode of FroNtier can be described by the following sequential steps:

- HTTP request encodes select statements;
- FroNtier reads the select statement of the HTTP request;
- FroNtier executes the statement in the database;
- FroNtier encodes and ships the result set to the client;
- HTTP reply encodes DB result sets in XML format.

These resulting XML query results are then cached using the HTTP caching server Squid, so that the same DB query, issued by many different clients, will be executed only once in the DB.

The COmmon Relational Abstract Layer software package (CORAL) [66], [67] is part of the LCG framework and, with its plug-ins, offers end users a set of standardized methods to access database sources such as Oracle, MySQL and SQLite. A FroNtier plug-in to CORAL [68] has been developed, which maps SQL queries to FroNtier HTTP requests, and extracts query results from the XML documents generated by FroNtier. This plug-in has embedded fail-over mechanisms that allow the user to specify multiple FroNtier servers, and cache servers to be contacted. Apart from being a read-only source, FroNtier acts in CORAL as any other data source.

Squid is a popular free proxy caching server that is mostly used to speed up web servers by caching repeated requests. Although it is primarily used for HTTP and FTP, it supports many protocols. Squid is highly configurable, provides extensive access control, a variety of cache sharing protocols, and several monitoring options. By deploying a central database, and a FroNtier server, one may easily deploy as many caching servers (Squids) as needed in any N-tier site to cache XML documents with database result sets. Squid can be setup in two different ways: as a normal proxy cache server working for a set of clients, or as a reverse proxy cache that lies in front of the server and acts as the server itself.

The overall view of the system is shown in Figure 4. The principal component is a server hierarchy that application clients contact. The server layer translates the client request into a data query, and returns to the client the desired information in a serialized form. The CORAL FroNtier plug-in receives the encoded object, de-serializes its contents, and delivers it to the client.

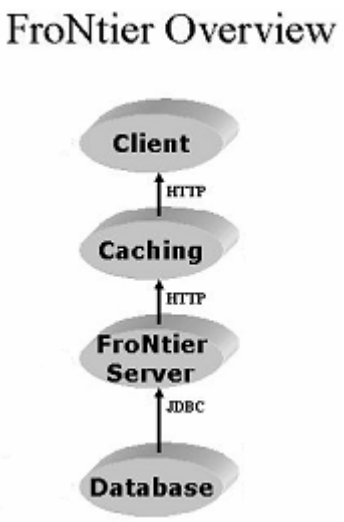


Figure 4 - FroNtier package overview

The key advantage of the FroNtier/Squid system is its lower administration requirement in comparison to a full database server deployment. As squid servers are easier to configure and maintain they may simplify the provision of a bcal database cache if human resources are limited.

Moreover, datasets used in physics computing for data analysis are generally small or medium, and the same datasets are typically accessed many times. This application characteristic makes the wide-are data caching a rather useful strategy.

Squid is very effective in providing read-only access to the static database information that FroNtier serves. Including a proxy-caching server layer in the system brings many advantages to a system like low latency, high scalability, ease of deployment, and maintainability.

One of the major disadvantages of the caching approach is that all database queries need to be performed upstream to the central database at least once. This can become a problematic bottleneck of the system. This is the reason why this database distribution solution works well with applications where the data is mostly read-only, and the queries repetitive. Repeated queries will always be found in the caching server, and will not result in real database queries.

Another major disadvantage of FroNtier is the cache consistency problems that may arise from its use. Once a query is cached in the Squid, there is no defined way to invalidate the cache in case the data is updated on the backend database. To overcome this problem one can setup a cache invalidation mechanism (see section 3.2.2), or define strict cache refreshing policies to avoid data inconsistency. In this case, as squid cache servers are not aware of database updates, applications running in a FroNtier/Squid environment need to be carefully designed to avoid possibly subtle consistency issues caused by stale cached data.

3.2.1 FroNtier deployment

A FroNtier deployment consists of a central database server that contains the data, the FroNtier server(s) near the database that serve the data to caches, and a distributed and typically hierarchal structure of cache servers (Squids). With this structure the clients access their local Squid cache. If the requested contents are not available there, the local Squid cache will automatically forward the request to the parent Squid cache or, in the case of the base

Squid caches, to the FroNtier server. The FroNtier server will serve the requested content to the Squids that will cache the reply, and serve it to the user that initially sent the request.

This solution offers very good performance, a simple deployment, but, as seen before, introduces some cache consistency issues.

The FroNtier production setup in LCG is shown in Figure 5.

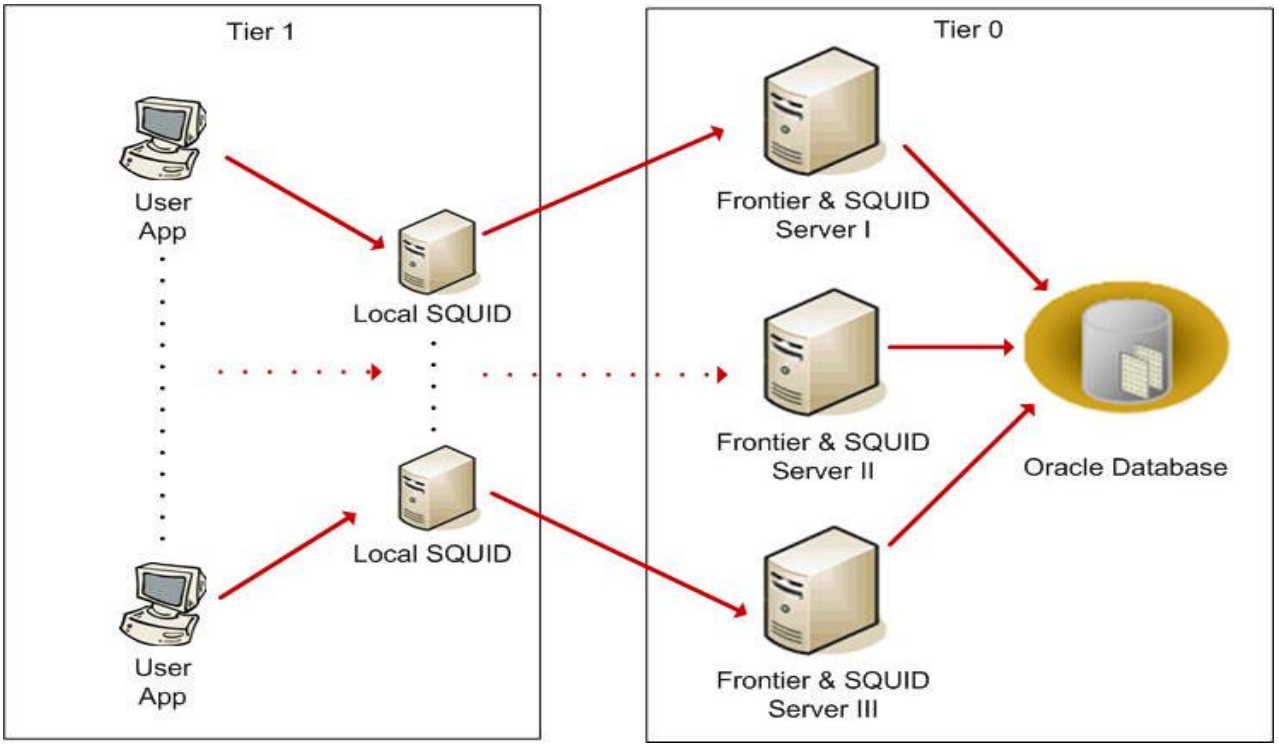


Figure 5 - 3D FroNtier/Squid production setup

The LCG Tier-0 setup consists of 3 load-balanced worker nodes running FroNtier that access the central database server which is a 4-node Oracle Real Application Cluster [69]. On these same 3 nodes there are 3 Squid servers running in reverse proxy mode. Several Squid servers will be deployed in N-tier sites to serve local user applications. It is expected that Tier-1 nodes are bigger and faster nodes as they will serve Tiers-2 caches.

3.2.2 Cache Consistency Issues

Several issues have been raised inside the community using FroNtier as whether the cache consistency offered by the FroNtier package suits the applications using it. This topic has received special attention as it is critical to the application stakeholders' choice of database distribution technique.

The cache consistency problem is that once a result set is cached in the Squid, there is no defined way to invalidate the cache in case the data is updated on the backend database. So, data consistency is not granted by the system and workaround methods have to be used to avoid this problem.

The different methods to avoid data inconsistency on a FroNtier setup are:

- define strict cache refreshing policies at application level;
- define cache refreshing policies at system level;
- setup a generic and automatic cache invalidation mechanism.

The following paragraphs describe each of these three options.

3.2.2.1 Strict cache refreshing policies at application level

This solution assumes that Squid cache servers are not aware of database updates and no action in taken directly on the cache servers to refresh its contents. These assumptions mean that applications running in this environment, and particularly its data models, need to be carefully designed to avoid possibly subtle consistency issues caused by stale cached data.

This solution uses the Squid feature where an application can issue a database request to the Squid forcing a data refresh. In these requests, the Squid cache does not use its cache contents but, instead, forwards the request to the FroNtier server that gets fresh data from the database.

The general policy defined is based on the concept of refreshed lookup tables. The application data models are based on lookup tables that are always refreshed (queries to these tables are always issued with the force refresh option). These lookup tables contain references to data tables that can be cached. The policy defines then that the lookup table records cannot be replaced or deleted, but only appended. This means that every time the database is updated, a new lookup record is created for the new data in the data tables while the old lookup records point to the old data values that may be cached.

The definition of this kind of strict policies for applications using controlled database updates on controlled database models is very difficult, especially in large software projects. For this reason, some functionalities that help implementing these policies were incorporated in the software framework (CORAL). CORAL provides an abstract interface that enables the client application to define refreshing policies at schema, table, or query level. It is the responsibility of the application to define the consistency policy by using this interface. This implies that the application must be designed specifically for FroNtier and would not work with other database access methods like direct Oracle access.

At a bottom line, these policies define a valid solution for the FroNtier data consistency problem with a very high impact on application development.

3.2.2.2 Cache refreshing policies at system level

A second approach to the cache consistency problem defines system level policies that do not impact the application development process but, based on cache cleaning actions, solve the problem in a system-wide perspective.

A first loose refreshing policy of this type solves the problem by setting server side scripts (on the Squid servers) that completely cleans the cache contents every static time interval. This way, no stale data is kept for more than the time interval defined.

Although this policy offers a very simple solution to a rather complex problem, it does not offer complete data consistency as when the database is updated, the cache servers will service stale data until the next cache cleaning action. Moreover, with this policy, data that was not updated will also be removed from cache, i.e., it will have to be requested to the database and cached the next time a client requests it.

A more strict approach to system policies defines that the cache cleaning scripts should only be executed when the database is updated, i.e., every time the database is updated, all the caches will be cleaned.

This solution offers complete data consistency but introduces other problems. If the database is updated to many times the cache servers will be overloaded with cleaning scripts.

This solution also introduces a synchronization problem as people doing the update may not be aware of the need to clean the caches. Additionally, the same cache can be used for different applications, which means the cache state will depend on other applications. Finally, this solution does not solve the problem of the data that is not updated being removed from cache.

3.2.2.3 Cache invalidation mechanism

A third approach to the data consistency problem is the development of a cache invalidation mechanism that invalidates individual cache objects based on the auditing of the database update operations. In this setup, cache servers are registered in the database, and changes in the database are propagated to the caches using cache invalidation messages. For example, if a database table is updated, all caches receive an invalidation message for all cached documents referring to the updated table.

In order to analyze the viability of a cache invalidation mechanism like this, a cache invalidation prototype for FroNtier was developed in the context of this thesis.

A cache content invalidation mechanism over a Squid, FroNtier, and Oracle database setup should be based on the following parts:

- Detect database changes;
- Propagate invalidation messages to Squid caches;
- Purge cached content in Squids.

Alternative implementations of each of these parts were analyzed.

Detect database changes

There are several ways to detect changes in an Oracle database. Four methods were identified: the view ALL_TAB_MODIFICATIONS of the Oracle data dictionary, database triggers, database auditing, and the Oracle Log Miner.

Oracle database triggers detect changes within an Oracle database. Triggers are callback functions associated with a set of firing conditions. Every time the trigger conditions are satisfied, the callback function is automatically executed by the database. This solution could be used to detect table updates and set the invalidation logic on a trigger call back function. Nevertheless, to setup the invalidation mechanism a database or schema wide update detection is needed. Database triggers can only be setup at table level, i.e., there is no database or schema wide update trigger.

The Oracle data dictionary view ALL_TAB_MODIFICATIONS contains all data modifications occurring in all tables of a given database. This view is not suitable for the prototype as it is updated off-line with up to three hours delay after the table modification.

The Oracle database offers a set of powerful auditing features. Using auditing functionalities, a complete table modification log can be obtained for a complete database. The setup of the auditing is rather straight forward, as a simple SQL command starts the log collection.

The Oracle Log Miner is an Oracle tool that also offers powerful auditing features by analyzing the database logs. This solution could suit the needs for an invalidation mechanism as it offers more information about database modifications, and introduces less performance overhead than the auditing solution. The significant disadvantage of this solution is its more difficult process of deployment, compared with the auditing solution.

Due to its simplicity and requirements fulfillment, the auditing solution was chosen to implement the detection of database modifications. The setup is completed with the use of a trigger on the log table. This trigger fires a procedure that starts the invalidation procedure.

Propagate invalidation messages to Squid caches

After detecting what tables are updated, and when are the tables updated, one needs to define a method to identify what Squids and what Squid objects should be invalidated due to a given database modification. This enables the communication of the table modification in the database to the Squid caches.

As Squid servers cache query - query result pairs and not table data, the major task to perform here is the conversion from a table name (modified in the database) to a cached object name, i.e., a query string. Two solutions were identified:

In the first solution, Squid servers using a given database are registered in the database. Every time a table is modified in the database, the database trigger function sends the table name to all Squid servers. Each Squid server receives an invalidation message containing a table name. In this setup, each Squid server scans its cache contents searching for queries on the modified table and purge them. This solution has the advantage of being simple and easy to deploy. On the other hand, it can considerably load the Squid server with the invalidation messages processing. Nevertheless, considering the application characteristics, the database update frequency is rather low. Additionally, at Squid server level, the typical bottleneck is the network bandwidth and not the CPU.

The second solution is based on FroNtier server instead. In this setup, the FroNtier server keeps the list of Squid servers, and is registered in the database to receive the names of the modified tables. For each query received from a registered Squid server, the FroNtier server logs the Squid server identification, the requested query, and the tables of the requested query. This logging could be done at FroNtier servlet level or using the apache logs on the FroNtier server. When the FroNtier server receives a database invalidation message with a table name, it sends an invalidation message to all Squids who requested queries with that table. This invalidation message, from the FroNtier server to the Squids, contains the list of all the queries requested by the destination Squid. These queries are stored in the logs on the FroNtier side, and, when received by the Squid server be immediately invalidated without further processing. This solution takes the processing of invalidation messages from the Squid servers to the FroNtier server but, instead, introduces a rather heavy overhead of logging and processing the invalidation messages to the FroNtier server CPU which is a typical bottleneck.

The base option here is to do the mapping between tables and queries in the Squid server (first solution) or in the FroNtier server (second solution). At a bottom line, the first solution is preferable for the main reason that is it simpler. The first solution would be based on an invalidation unit running on each Squid server that handles most of the invalidation work.

The bigger challenge of this setup is the complexity of maintaining the invalidation unit code on all Squids deployed over the grid. This number is rather large, for example, the current FroNtier setup for the CMS experiment is composed of 32 Squid servers spread around Tier 1s and Tier 2s centers of the LCG.

Apart from the chosen solution there are some issues common to both solutions. When a table is modified, all queries on that table are invalidated. Nevertheless, when a table is updated, not all previous queries on that table are necessary invalid, for example, queries on

data not updated of the modified table. In these cases, there is always the danger of invalidating objects that are still valid (over-invalidation). This is one example of why the invalidation procedure can be tricky, and why the invalidation rules should be detailed analyzed before implementation.

Another issue that must be taken into account is the necessity to register the Squid cache hierarchy, so that the second level Squids (using other Squids as backend) also receive invalidation messages. Alternatively, a system for sharing invalidation messages between Squids could be setup.

Purge cached content in Squids

The last step of the invalidation process is the actual purge of the Squid objects from the cache. This is a rather straight forward process that can be accomplished in two ways: the HTTP purge command that can be issued externally to any Squid cache, or the Squid purge tool that has to run locally on the Squid server. The HTTP purge command can only purge one object at a time while the Squid purge tool uses regular expressions for purging multiple objects with one command. As the installation of the Squid purge tool could be integrated in the deployment of the invalidation unit, it is considered the best option between the two.

3.2.2.4 Cache Consistency Summary

This section presented the data consistency issues of the FroNtier/Squid project and described three possible approaches to handle these issues.

FroNtier is not able to replace a database access method with exactly the same consistency semantics as a direct database connection, except with the implementation of an invalidation mechanism. Generally, FroNtier/Squid does not offer data consistency warranties by it self. Conversely, the projects using it do not require 100% of data consistency, although it would be preferable. Therefore, applications running on top of FroNtier/Squid setups will always have to be aware of the consistency issues in order to maintain the desired level of data consistency.

The discussion on the trade off between data consistency level and data consistency maintenance complexity is still on going. Policies that offer "sufficient" data consistency for the application have to be defined, taking into account the application access/update sequence of the application. In the LCG, most of the application data will be updated, rather than replaced.

Finally, this brief discussion on the setup of an invalidation mechanism for FroNtier showed the viability of the project, and identified the different issues that have to be taken into account when developing such a solution. If such a prototype was to be developed, it could be tested using the tools developed for this thesis, and the performance overhead introduced by the invalidation mechanism could be measured.

Chapter 4 - Performance Analysis Methodology

Performance analysis is a key step in the design of new computer systems including processors, languages, operating systems, networking architectures, user applications, or database systems. Several techniques are used in performance evaluations such as statistics, probability theory, experimental design, measurement, simulation, queuing theory, data analysis, and data presentation.

Performance evaluations are required at every stage of the life of a software system, including its design, development, sales, use, upgrade, etc. Evaluations are typically needed to compare alternative software systems or, in cases where no alternative exist, to evaluate the current system to identify what components of the system are the bottlenecks, and, with this information, reconfigure, tune, or upgrade the system. The different alternatives for database distribution presented in the previous chapter are not directly comparable as they use different replication schemas that offer different levels of latency, data consistency, and redundancy. Nevertheless, evaluating each of the alternatives can provide useful insight on the technology, for example, by finding bottlenecks or by identifying the optimal configuration.

This chapter will make a brief overview of performance evaluation terminology and techniques while exposing the methodology used in the performance evaluation of FroNtier.

4.1 Methodology definition

Several performance analysis methodologies exist, such as measure the performance of a system under a real workload, simulate a workload measuring the system behavior, or build an analytical model of the system.

Performance evaluations commonly consist of monitoring a system while loading it with a given workload, and, from these measurements, build a model of the system. An analytical model can be built from scratch without measurement in order to predict the performance of the system.

The most common method to compare the performance of two systems is called benchmarking. This technique consists of analyzing the behavior of the systems under specific designed workloads (benchmarks).

As there is no standard measurement environment or technique for the wide variety of software systems, the base steps of a performance evaluation are to select the best evaluation technique, the best performance measures, and the best measurement environment.

Typical problems occurring in performance evaluations are: specifying performance requirements, evaluating design alternatives, comparing two or more systems, determining the optimal value of a parameter (system tuning), finding the performance bottleneck (bottleneck identification), characterizing the load on a system (workload characterization), determining the number and size of components (capacity planning), or predicting the performance at future loads (forecasting).

Based on R. Jain [70], we can define a performance evaluation methodology composed of the following steps:

Define the system and state evaluation goals

The first step in the methodology is to define the system to be tested. Without understanding the system and its boundaries one is unable to evaluate it. After defining the system, the analyst should define the goals of the evaluation. One of the most common reasons of failure of a performance evaluation is an unclear definition of the goals of the evaluation. The goals statement should be clear and unbiased, i.e., should be easily understandable and not based on results expectations.

List services and outcomes

If we look at the system under evaluation as a service provider, we can identify the services it provides. For example, a database system provides services such as: query tables, create tables, populate tables, etc. This list of services and possible outcomes of each service is the base of the evaluation methodology. It is used in the selection of metrics, parameters, workloads, etc.

Select metrics

Metrics are the criteria used to evaluate the performance of a system, for example, the response time or the throughput. Metrics should be representative of the system behavior, and relevant for the problem.

Select parameters

Parameters are configurable elements of the system. System parameters are parameters of a given setup of the system. Workload parameters are parameters that vary with user requests. It is quite important to identify the complete set of parameters of the system under evaluation.

Select factors and their values

Factors are selected parameters used as variables on the evaluation. A performance evaluation is commonly based on the study of the impact of the variation of the factors on the system. The correct use of parameters and factors is another critical point in an evaluation. Not all parameters have an equal effect on the performance, some parameters maybe used as factors while others may be fixed at their typical values. It is very important to identify the parameters which, if varied, will have significant impact on performance of the system.

Select evaluation technique

Evaluation techniques are: measurement, simulation, or analytical modeling. Different evaluation techniques have different speed, accuracy, cost, etc.

Select workload

Workloads are the requests made by the user to the system, for example, the queries and other requests the user executes on a database. Workloads define how the system under test will be exercised. All results depend on the correctness and representativeness of the workloads used. Moreover, a workload can exercise different services with different levels of detail, and can have different levels of repeatability.

Design experiment

The experiment design defines a set of experiments that aim to provide most information with least effort. The definition of the experimental design based on existing techniques help

organize a set of measurement or simulation experiments to obtain maximum information with minimum number of experiments. A precise experimental design uses the correct level of detail (right number of measurements and parameter values) to separate the effects of individual factors.

Analyze and interpret data

After executing the experiments, a precise analysis of the gathered data is fundamental. A common mistake is to not actually analyze the data taken and, instead, collecting a lot of data, and doing no analysis. While analyzing data one should make sensitivity analysis, analyze outliers, analyze measurements variability, validate measurements, and take into account system details like caching effects, buffer sizes, monitoring overhead, testing conditions, etc.

Present results

The last part of the evaluation is the presentation of the conclusions. Effective result presentation to project stakeholders and managers can determine the success of a performance evaluation. This step is commonly under valuated and a lot of unsuccessful cases can be explained by wrong result presentation. Result presentation should be concise, should use graphical data as much as possible, and should report on assumptions and limitations of the evaluation.

At a bottom line, an evaluation should be based on problem definition and result presentation instead of using a model as an end in itself.

This list of steps defines a systematic approach that can help getting to accurate conclusions and avoiding typical errors in performance evaluations. The next sections describe the process of using the defined methodology and discuss all the listed topics in detail.

4.2 Goals statement

The main objective of this thesis is to elaborate a systematic performance evaluation of the FroNtier package. The database caching solution offered by FroNtier can be exercised in multiple ways. The methodology defined in the previous section will be followed for this FroNtier performance evaluation.

Before executing a performance analysis one should clearly understand the problem. The following questions help understand the FroNtier problem:

- How fast are the individual components of the system (Database, Application Server, Cache Server and Network)?
- What impact has different data (content, size, storage type, compression), different complexity database schemas, or different caching policies in the performance of the system?
- How do database throughput, network bandwidth, payload size, number of clients, or server CPU usage correlate?
- How can performance bottlenecks be identified in such a complex software stack?

Additionally to the performance evaluation resulting from this testing activity, a benchmark for FroNtier servers will be developed so that it can be reused in future

evaluations or comparisons. This benchmark is supposed to be independent of the application and of the database schema in use.

In a lower level, the objective of this evaluation is to identify performance bottlenecks of a given FroNtier setup in an easy manner. Moreover, the performance evaluation should include the complete software stack involved in the FroNtier package, i.e., the CORAL / FroNtier plug-in, the FroNtier Client, the Squid servers, and the FroNtier Servlet (see section 3.2 for more details).

Based on these goals, the following sections will describe the methodological steps for developing FroNtier tests.

4.3 List services and outcomes

If we look at the system under evaluation as a service provider, we can identify the services it provides.

As see in section 3.2, FroNtier is a rather simple system. The services provided by a typical database system are, for example, query table, create tables, populate tables, etc. FroNtier is a slightly different system primarily because it is a read only system, i.e., write operations are not part of the FroNtier services.

In a general way, the base service offered by FroNtier is the ability to select different columns in different tables in any schema of a specific Oracle database. The ability to access the data through Squid caches or directly from the FroNtier server in a controlled manner could be defined as a service. In this evaluation, this feature will be considered a parameter as the workloads exercised will be the same for FroNtier server and for Squid caches.

The outcomes of the base service of FroNtier are always XML files containing the query results. This XML file is transmitted over an HTTP connection and can represent the following outcomes: a valid result set with the correct answer to the query, a valid result set with the wrong answer to the query, or an error code and respective message.

4.4 Metrics selection

Performance metrics are the criteria used to evaluate the performance of a system, for example, the response time (the time to serve a request), or the throughput (the number of transactions per second).

After listing the services and outcomes of the system, we should define metrics for each of the services provided by the system. A number of metrics should be selected to separately measure the speed, the reliability, and the availability of each of the services.

The speed is measured when the system can execute the requested job correctly. Examples of speed metrics are the response time, the response rate (throughput), and the resource utilization. These metrics are known respectively as responsiveness (time to do the job), productivity (number of jobs per unit of time), and utilization (percentage of time the resource of the system was used).

The reliability is measured when the system works but executes the job with errors or incorrectly. It is important to define all the different types of errors.

The availability is the ratio between the time the system is working normally (uptime) and the time the system can not execute user requests (downtime). As with errors, the different

downtime modes should be classified and measured separately. For example, a system can be down due to network failure or software failure.

In systems with many users, special attention should be given to differences between individual metrics and global metrics. Individual metrics are metrics for each user, while global metrics reflect the system-wide perspective. For example, availability is a global metric, while throughput can be either global or individual. Optimizing a system for a global metric is different from optimizing it for individual user metrics. A typical case is the individual throughput of a user that can be easily optimized by compromising the global throughput. In this case, system-wide throughput, and its distribution among all users should be studied.

After selecting the metrics, some of them can be identified as potential interesting for the analysis. For example, the ratio between two metrics that do not vary between them can be used instead, for example, the ratio between throughput and response time.

Some commonly used performance metrics are:

- the response time is the time between the user request and the system response. It can be defined as the time between the user request and the beginning of the response, or between the user request and the end of the response. The end of the response should be considered in the case where the response is long (for example in a database system);
- the variability of the response time is usually a very important metric as it tells whether the system is performing constantly;
- the reaction time is the time between the submission of the request, and the beginning of the processing. To measure the reaction time the system must be monitored;
- the stretch factor is the ratio between the response time and the load. Usually the response time increases with the load on the system;
- the throughput is the rate (requests per unit of time) at which the requests can be served by the system. The throughput is measured in jobs per second on batch systems, in packets per second on networks, in transactions per second on database systems, etc.
    - the throughput of a system generally increases as the load of the system initially increases; after a certain load the throughput stops increasing, and may start to decrease. The maximum achievable throughput under ideal workload conditions is called nominal capacity of the system. For networks, for example, this nominal capacity is called bandwidth.
    - Often, the response time at maximum throughput is too high to be acceptable. In such cases, it is more interesting to know the maximum throughput achievable without exceeding a pre-specified response time limit. This may be called usable capacity. In most application the optimal operating point is the point where the throughput reaches its maximum where the response time is still good: this is the point where the response time increases rapidly in proportion to the load on the system, and the throughput gain is small. Before this point the response time does not increase significantly, but the throughput rises as the load of the system.
    - It is also common to measure capacity in terms of load, for example, the number of users rather than the throughput;
- the fairness of the distribution of the throughput; as for response time, the variability of the individual throughputs should be analyzed;

- the efficiency is the ratio between the maximum achievable throughput (usable capacity) with the nominal capacity; a throughput of 85Mbps in a 100Mbps connection denotes a connection with 85% of efficiency;
- the utilization of a resource is the ratio of time a given resource is busy servicing requests. It may be the usage of resources such as CPU, disk, memory, network, etc. The resource with the highest utilization is called the bottleneck. Performance optimization on this resource will have the highest payoff. Finding the utilization of various resources is one of the most important parts of a performance evaluation, i.e., system monitoring (see section 4.10.2);
- the reliability is usually measured as the probability of errors, or the mean time between errors;
- the availability is usually measured as the Mean Time to Failure, which is the mean uptime;
- a famous metric is the cost/performance ratio where the performance can be defined as one of the metrics above, for example, the global throughput.

For this performance evaluation, the following metrics were selected:

- the **global throughput** of the system is the base performance metric used in this study – the throughput is measured not in transactions per second (as typically done in database systems), but instead in data bytes per second, so that the obtained values are independent of the transaction size;
- the **response time** - the request processing time (reply time) is included;
- the **variability of the individual and global throughput**;
- a slightly adapted **stretch factor**, i.e., the ratio between the throughput and the load of the system;
- the **utilization** of resources – CPU consumption, memory usage, disk space, and network bandwidth for clients, FroNtier servers, Squid servers, and database servers. This metric is of major importance for bottlenecks identification, and will be further discussed in section 4.10.2;
- the **reliability** - the probability of errors (typically occurring under heavy load);
- the cost/performance metric will not be directly used, but the performance impact of the hardware resources (for example, FroNtier servers or Squid caches) will be analyzed.

Conversely, some metrics are not considered useful and will not be used. The variability of the response time is assumed to be the same as the variability of the throughput and therefore is not used. The reaction time will not be used as its values for FroNtier are typically insignificant. Moreover, these values are mainly dependent on network latencies between system components. As very low response times are acceptable in FroNtier setups, the nominal capacity is the same as the usable capacity. For this reason, the efficiency (ratio between the usable capacity and the nominal capacity) will not be used in this evaluation. Finally, the availability will not be used as it is a metric commonly used for analyzing systems in production, which is not the case.

4.5 Parameters selection

Parameters are configurable elements of the system. It is quite important to identify the complete set of parameters of the system under evaluation. If a parameter is not taken into account, it can introduce significant errors in the analysis. There are two types of parameters: system parameters or workload parameters.

System parameters are parameters of a given setup of the system under study. Their values can be tuned by system administrators. Examples of system parameters are the number of FroNtier servers of a given setup, or the value of some operating system parameter in the FroNtier server.

Workload parameters are parameters that vary with user requests. Their values are controlled by the end user or end application. Examples of workload parameters are query sizes, or FroNtier documents compression level.

We start by identifying all types of parameters of a FroNtier system. Along with the list of parameters, we define, for each parameter, its type (categorical or quantitative), its possible values (levels), and its typical value (the value used in experiments if the parameter is not selected as an experiment factor).

In a FroNtier setup, the identified system parameters are:

- number and configuration of FroNtier, Squid, and database servers (memory, number of CPUs, disk, etc.):
    - these three groups of parameters are all quantitative;
    - possible values depend on available technology and budget. In our particular case, only the number can be changed as 2 servers are available;
    - except the number of FroNtier/Squid servers that as a typical value of 1, these values will not be changed during the experiments (section 5.1 describes the hardware setup used in this evaluation).
- servers network connections speed:
    - quantitative (bits per second);
    - 10Mbps, 100Mbps, or 1Gbps;
    - 100Mbps (1Gbps is not available for testing).
- FroNtier server distribution:
    - categorical;
    - all FroNtier server distributions. FroNtier server distributions used in this evaluation were versions 3.1 and 3.3. Version 3.1 introduced FroNtier documents compression. Version 3.3 included a feature that using the HTTP keepalive feature reduced error rates for high load tests;
    - FroNtier server version 3.3.
- FroNtier client distribution (it is considered a system parameter as it works together with server distributions):
    - categorical;
    - all FroNtier client distributions. The FroNtier client distribution used was the one directly associated with the distribution of the FroNtier server: versions 2.4 and 2.5;
    - FroNtier client version 2.5.
- CORAL distribution (as the FroNtier client, it works together with the FroNtier server distributions):

  - categorical;
  - all CORAL distributions. CORAL 1.3, CORAL 1.5, and CORAL 1.6 were used. CORAL 1.3 changed the mapping between database data types and CORAL data types (data sizes are different). CORAL 1.5 introduced compatibility with FroNtier documents compression;
  - CORAL 1.6.
- Squid distribution:
  - categorical;
  - all Squid distributions;
  - Squid cache version 2.5.
- FroNtier servlet configuration – maximum active database connections in FroNtier servlet:
  - quantitative;
  - any positive number (0 means no limit). The sum of the values of this parameter in all FroNtier servlets connecting to the same database user must be lower than the database's maximum number of sessions per user;
  - 10.
- Squid configuration - cache size:
  - quantitative (bytes);
  - 0 to maximum cache size (in this case, 150GB of disk);
  - 0 (empty, or nearly empty caches).
- Squid configuration - cache peering (Squid peers can share cache content):
  - categorical;
  - yes/no: use cache peering between Squid servers or not;
  - no, as only one Squid server will be typically used.
- Squid configuration - cache hierarchy:
  - categorical;
  - yes/no: use Squid cache hierarchy or not;
  - no, as one Squid server will be typically used.
- server's operating system parameters - TCP window size:
  - quantitative (bytes);
  - 0 to the maximum size specified in /proc/sys/net/core/wmem_max and /proc/sys/net/core/rmem_max (typically 128kB);
  - 64kB.
- database parameters (total database size, query caching, memory structure's sizes, max sessions per user, etc.):
  - quantitative;
  - hardware dependent;
  - section 5.1 describes the database setup used on the experiments.
- number of client nodes:
  - quantitative;
  - between 1 and 15 (client nodes available for these tests);
  - 10.
- location of client nodes:
  - categorical;
  - any computing center in the LCG;

    - CERN tier-0.
- configuration of the client nodes:
    - quantitative;
    - depends on the technology and budget;
    - section 5.1 lists the characteristics of the client nodes used in the experiments.
- number of test clients running in each client node:
    - quantitative;
    - any positive number;
    - 2.

Complementarily, the workload parameters are:

- Number of queried tables:
    - quantitative;
    - any positive number (typically small);
    - 1.
- Number of result columns:
    - quantitative;
    - 1 to 1000;
    - 1.
- Number of result rows:
    - quantitative;
    - depends on database distribution;
    - 10000.
- Number of columns in each queried table:
    - quantitative;
    - 1 to 1000;
    - same as number of result rows.
- Number of rows in each queried table:
    - quantitative;
    - depends on database distribution;
    - same as number of result columns.
- C++ data types (database column types):
    - categorical;
    - int (NUMBER(10)), float (BINARY_FLOAT), double (BINARY_DOUBLE), std::string(x) (VARCHAR2(x)), and coral::Blob (BLOB);
    - std::string(100).
- Size of records in table:
    - quantitative;
    - depends on database distribution;
    - 40 bytes.
- Query data:
    - categorical;
    - real application data, simulation data, or random data;
    - random data.
- Query data compressibility (randomness):

    - quantitative;
    - 0 to 100% (relation between the network data and the database data);
    - 90% (network data is 90% the size of the database data).
- Query frequency or client delay time between queries:
    - quantitative (seconds);
    - any number;
    - 0 seconds.
- FroNtier documents compression factor:
    - categorical;
    - between 0 and 9, where 0 represents no compression, 1 minimal compression, and 9 maximum compression;
    - 5: medium compression.
- database access method:
    - categorical;
    - direct FroNtier access, Squid cache access, access to Squid but forcing cache refresh on each query, and direct database access;
    - direct FroNtier access.
- CORAL client connection mode:
    - categorical;
    - connect on every query and connect only once;
    - connect only once.

4.6 Factors selection

Factors are selected parameters used as variables on the evaluation. This performance evaluation studies the performance impact on the system of the variation of the factors.

All software systems have a huge number of parameters. Nonetheless, not all parameters have an equal effect on the performance; some parameters can be used as factors while others may be fixed at their typical values. It is very important to identify the parameters which, if varied, will have significant impact on performance of the system. The values that a given factor can take are called levels. The levels of a given factor are a subset of all the possible values of the parameter associated with the factor.

In this section, we proceed with the discussion on which parameters will be used as factors. Workload parameters and system parameters can be equally chosen as factors. First experiments will use a small number of factors (primary factors) and levels, while later experiments will use a expanded number of factors (secondary factors). So, from the list of parameters defined in the previous section, a list of primary and secondary factors and respective levels was built.

The parameters chosen as primary factors are:

- number of clients – it is derived from other parameters. It is calculated as follows:

```
number of clients = number of nodes * number of clients in each node
```

    - The levels are 1, 2, 5, 10, 20, 50, 80, and 100;
- query size (as seen on the database) - it is derived from other parameters. It is calculated as follows (record size as seen in DBA_USER_TABLES.AVG_ROW_LEN after computing table statistics):

```
query size = number of result rows * number of result cols * record size
```

    - The levels are 1kB, 10kB, 100kB, 1MB, and 10MB;
- FroNtier documents compression factor:
    - levels are 0, 1, 5, and 9;
- data access method:
    - levels are direct FroNtier access, Squid cache access, access to Squid but forcing cache refresh on each query, and direct database access.

The parameters chosen as secondary factors (listed by order of importance) are:

- number of queried tables:
    - Levels are 1, 2, 5;
- FroNtier server distribution:
    - levels are version 3.1 and version 3.3 (to analyze keepalive function impact);
- location of client nodes:
    - levels are CERN tier-0 and FNAL tier-1;
- CORAL client connection mode:
    - levels are connect on every query and connect only once;
- number of FroNtier/Squid severs:
    - levels are 1 and 2;
- maximum active database connections in FroNtier servlet:
    - levels are 5, 10, 20;
- number of rows in each queried table:
    - levels are same as number of result rows and 10 times that number;
- C++ data types (database column types):
    - levels are int (NUMBER(10)), float (BINARY_FLOAT), double (BINARY_DOUBLE), and std::string(x) (VARCHAR2(x));
- query data:
    - levels are random data and simulation data;
- query data compressibility:
    - levels are 5, 30, 50, 70, 90;
- query frequency:
    - levels are 0, 10, 30, and 60;
- squid configuration:
    - cache size, cache peering, cache hierarchy.

The number of database servers is not possible to be elected as a factor as only a limited number of servers is available for testing. Excepting the maximum active database connections in FroNtier servlet, all nodes and server parameters (test clients, FroNtier, Squid, and database) are tuned for their typical values, and are only modified if a test result indicates a better value for a specific parameter. The speed of the network connections between nodes and servers is fixed at 100Mbps as higher speeds are not available. FroNtier client and CORAL distributions are used in accordance with the FroNtier server distribution in use. There is no identified reason to test different versions of Squid. The number of columns in each queried table is supposed to have a very small impact on the performance on the system, therefore, was set with its typical value.

4.7 Evaluation technique selection

A rather base step on a performance evaluation is to select the evaluation technique to be used. The performance techniques most widely used are: measurement, simulation, and analytical modeling.

Measurement consists of monitoring a system while loading it with a given workload. To measure the performance of a software system two tools are needed: one to load the system (load driver), and another to measure the results (monitor). These elements will be discussed respectively in sections 4.10.1 and 4.10.2.

Analytical modeling consists of using an abstract model of a system to analyze its behavior. A model must be designed to be similar to the real system in structure and behavior. An analytical model defines a system in terms of its components, and a mathematical or logical specification of those components. The analyst can then take conclusions about the model analytically, for example, using mathematical procedures to reason about the specification. Two key steps in analytical modeling are the processes of validate and verify the built model. Validation makes sure the model is the adequate for the defined purpose, while verification makes sure the model was well built.

Simulation consists of building a simulator that imitates the behavior of the system under study. Simulations are based on models, but go a step further on the details obtained by developing a simulator that computes the model. A simulation can be understood as a model animation. To execute a simulation, often the mathematical model is translated to a computer program, and the simulation consists of running the program with controlled inputs. The analysis is done over the output of that computation.

In order to decide what technique to use one can take into account several considerations such as the life-cycle stage of the system, the time available for the execution of the evaluation, the accuracy needed on the analysis, the costs, etc.

For example, if a given product is in design phase, it is not possible to make measurements of the system. If there is not enough time to do measurements (usually the slowest technique), a simulation or an analytical model should be used instead. Conversely, if a big level of accuracy is needed, measurement techniques should be applied, as they offer the biggest accuracy.

As FroNtier is still on validation phase, no real production environment exists, and the time and cost of the evaluation are not real constraints, the measurement of the system is the obvious option for the evaluation technique.

4.8 Workload selection

Workloads are the requests made by the user to the system, for example, the queries and other requests the user executes on a database. The terms benchmark and workload are used synonymously: the process of comparing the performance of two or more systems by measurements is called benchmarking, and the workloads used in the measurements are called benchmarks.

There are two types of workloads: real workloads and synthetic workloads. A real workload is the workload of a system in normal operation; it usually can not be repeated, and thus, it can not be used as test workload. Nevertheless, in some cases the real workload can be captured, and used later as test workload. A synthetic workload has similar characteristics

but it is produced artificially, so that it can be repeated. Synthetic workloads can typically be modified easily and can include features not seen in real workloads.

The workload selection is the most crucial part of any performance evaluation project. It is easy to take misleading conclusions if the workload is not properly selected. If a workload model is built, the effect of changes in the system can be studied in a controlled manner by varying the parameters of the model.

When defining the workloads, the major considerations to take into account are: the services exercised by the workload, the level of detail, the representativeness of the workloads, the timeliness, the loading level, the impact of external components, and the repeatability.

If we look at the system as a service provider, as seen in section 4.3, we can base the workload definition in that list of services. In our specific case, FroNtier is defined as a single service system: the ability to reply to database queries, i.e., to retrieve different columns in different tables in any schema of a specific Oracle database.

The level of detail at which the workload exercises the service can go from exercising all types of requests to only exercise the most common requests. Alternative approaches exist, such as, associate a frequency to each request type, and exercise each request the percentage equivalent to that frequency. Another alternative is to collect a time-stamped sequence of requests in a real system and use it as test workload. In FroNtier, different types of queries to the database have to be exercised, for example, big/small queries, single/multiple table queries, single/multiple column queries, different column types, etc.

A test workload should be representative of the real application in terms of arrival rate of the requests, resource exercise, and resource usage. Although there is still no real application running FroNtier, this evaluation will try to do some approximations to the expected real FroNtier workloads. The arrival rate of the requests can be easily controlled through the query frequency used by the test client. Concerning the resource exercise, FroNtier offers a single service that uses all resources on the system, i.e., the FroNtier servers, the Squid servers, the FroNtier and Squid servers' elements as disk, memory, CPU, and network. The level of resource usage can be easily controlled with the number of nodes, number of clients in each node, size of the query, etc.

Workloads should simulate the usage patterns of real applications as much as possible. The sequence of service requests is usually a very important factor on building the workload. Although the FroNtier server is a stateless system, the Squid caches are not. As the requested queries to the Squid caches may, or may not, be cached, the order of arrival of the requests is a significant factor. In this evaluation, the test clients will generate synthetic workloads that will control, on the client side, whether the queries are cached or not. On the server side, the queries will always be cached except an explicit refresh is requested by the client. The assumption is that, in terms of performance, there is no difference between a Squid cache miss (requested object not found on the cache) and a Squid cache forced refresh (client requests a cache refresh). In fact, a Squid cache miss is slightly slower than a Squid cache forced refresh as it includes a failed cache object matching.

The loading level at which the workload will exercise the system should be taken into account when defining workloads. A workload can exercise a system to its full capacity, beyond that capacity, or to the real level of load of the system. In this evaluation, the

objectives take us to the analysis of stress conditions, i.e., identification of the full capacity of the system by analyzing its behavior in the transition between full capacity and beyond.

Another important aspect is the impact of external components on the performance of the system. Different workloads get different impact from external components. The network bandwidth available can be considered an external component with great influence on the system. For example, a workload that exercises queries on big database tables depends more on the network bandwidth available that a workload exercising queries on small database tables: this has to be measured and taken into account.

Workloads should be easily repeatable and give results with low variance. A workload that makes random demands on resources will need more runs to get meaningful results. Real user systems are most of the times not repeatable. The analysis of the real-user environment and the capture of its key characteristics is called workload characterization. Several statistical techniques can be used for workload characterization such as averaging, dispersion analysis, histograms, principal-component analysis, Markov models, clustering, etc. All workloads used in this evaluation will be highly repeatable as they are synthetically generated. This implies a careful analysis of the results variability.

Three workloads are used in this evaluation:

- CORAL - a purely synthetic workload that exercises the CORAL/FroNtier Plug-in API with simple queries to a test database structure;
- COOL - based on the COOL API usage patterns;
- ATHENA – a simulation application called Athena

These workloads are shown as a set of user transactional scripts, i.e., queries to the system and on the Athena workload, some computation on the client side. All the workloads are executed by the same load driver. This load driver consists of a client program that reads a client script with a executable or a list of database queries (see section 4.10.1 for more details).

The CORAL workload is a purely synthetic workload exercising the CORAL/FroNtier Plug-in API with simple queries to a test database structure. This workload consists of a list of queries based on the variation of the workload parameters selected as factors in section 4.6. Together with this list of queries, a database creation script is developed to create the database against which the queries can be executed.

The following query is an example of a query of this workload:

```
SELECT string FROM T_STRING where rownum < 10000;
```

The table T_STRING is a single column table with 10000 rows of varchar2(100) filled with the function `dbms_random.string( 'X', 100 )` that generates random strings of 100 characters, resulting in an average record size of 104bytes (4 bytes of row header).

The factors and respective levels used in this example are:

- query size: 10000 * 1 * 104 = 1MB;
- number of queried tables: 1
- number of rows in each queried table: 10000;
- C++ data types (database column types): std::string(100);
- query data: random data.

We would then need to define in the load driver the remaining workload parameters, i.e., the FroNtier document compression factor, the database access method, CORAL client

connection mode, and the query frequency. As from this example we could vary any factor to any other level, this workload defines the base for the experiments described in the next section.

The second workload is based on the usage patterns of the COOL API. The queries used in this workload are captured from a typical utilization of the COOL API. The logs of an application using the COOL API were captured, and the queries related to COOL were retrieved. As the COOL API uses the CORAL layer, this workload will exercise both the CORAL and the COOL layers. It consists of a list of COOL queries that are executed (using CORAL) against a COOL database structure filled with random data. The following query is an example of a COOL query:

```
  SELECT "OBJECT_ID", "CHANNEL_ID", "IOV_SINCE", "IOV_UNTIL",
"USER_TAG_ID", "SYS_INSTIME", "ORIGINAL_ID", "NEW_HEAD_ID", "X0"
  FROM IOV_F0001_IOVS
  WHERE ( ( ( IOV_SINCE <= 0 )   AND ( 0 < IOV_UNTIL ) ) OR  ( ( 0 <=
IOV_SINCE )   AND ( IOV_SINCE <= 0 ) ) )
  ORDER BY CHANNEL_ID ASC, IOV_SINCE ASC
```

Here we can see that this workload explores much more varied query types. Nevertheless, this workload is used for a much limited analysis of specific factors (see section 5.2.11 for more details).

The Athena workload consists of a simulation program that has a very similar behavior to the final data analysis programs in terms of data access pattern and data processing on the client. When using this workload, each client will run an Athena job that accesses a specific database structure and processes the queried data. These simulation jobs use the COOL API and so, the CORAL layer to access the database, i.e., all the software stack is used in this workload. So, this workload will put more load on the client CPU and the database access pattern will be very sparse compared to the other workloads used in this study. Section 5.2.13 describes the experiment with this workload).

4.9 Experiment Design

In order to take meaningful conclusions for the different levels of the selected factors, it is very useful to isolate the measurement of the impact that each factor has on the performance of the system, and, additionally, to analyze the performance effects between factors, i.e., how the impact on performance of a factor is influenced by the other factors. In this last case, it is also interesting to see if the variation of a single factor reflects a performance variation, or if this performance variation is introduced by random variations of uncontrolled parameters.

For these purposes, there are several experimental design techniques that help organize experiments to obtain maximum information (most effective data analysis) with least effort (the minimum number of experiments). An experiment design consists simply on the number of experiments, the factor level combination for each experiment, and the number of repetitions of each experiment. This section briefly describes the three options for the experiment design.

Simple Design

This naive experiment design follows the idea of "change one thing at a time". It bases all the experiments in a typical configuration of the system, and then varies a factor at a time to see how that factor impacts performance. This technique only tries all levels of a factor against the typical value of the others.

This design may easily lead to wrong conclusions as the effect of one parameter may depend on the other one. The following experimental designs are considered better.

Full Factorial Design

This design explores all possible combinations of all levels of all factors. The advantage here is that every possible combination is tested. All interactions between factors can be found. The problem introduced by this design is the time cost involved in executing all possible combinations. Typically, this design technique defines a number of experiments too large that can be reduced by:

- reduce the number of levels of each factor;
- reduce the number of factors;
- use fractional factorial design (see below).

A full factorial design that uses two levels for each factor needs 2k experiments. This is a very popular design called 2k design.

Fractional Factorial Design

While the full factorial design crosses all levels of each factor with all others, in the fractional factorial design, only some levels of some factors are crossed with others. This drastically reduces the cost of the experiments. On the other hand, it introduces some complexity in the analysis, and some interactions between factors may be lost.

The experiment design methodology for this evaluation is based on the following steps:

- validate method with experiments that use few factors and few levels (2k design);
- proceed to experiments using large number of factors but a limited number of levels each. This helps determining the relative effect of the various factors and sorting out the factor impact importance (2k design);
- continue with experiments using the simple design. This will allow us to do a basic analysis of the impact of a large number of individual factors and its numerous levels;
- finish with detailed experiments using few factors but a lot of levels. Here a full factorial analysis is used.

The details of each experiment are documented in chapter Chapter 5 -.

4.10 The Test Framework

In order to accelerate the execution of the experiments, a test framework was developed. It automates most of the experiment execution and data gathering. It is composed of a load driver and a monitoring unit. Additionally, the load driver is integrated within a test control unit that, besides managing the execution of the client emulators, includes an integration program that gathers client and monitoring data into a single analysis point.

The following figure shows the base architecture of the test framework:

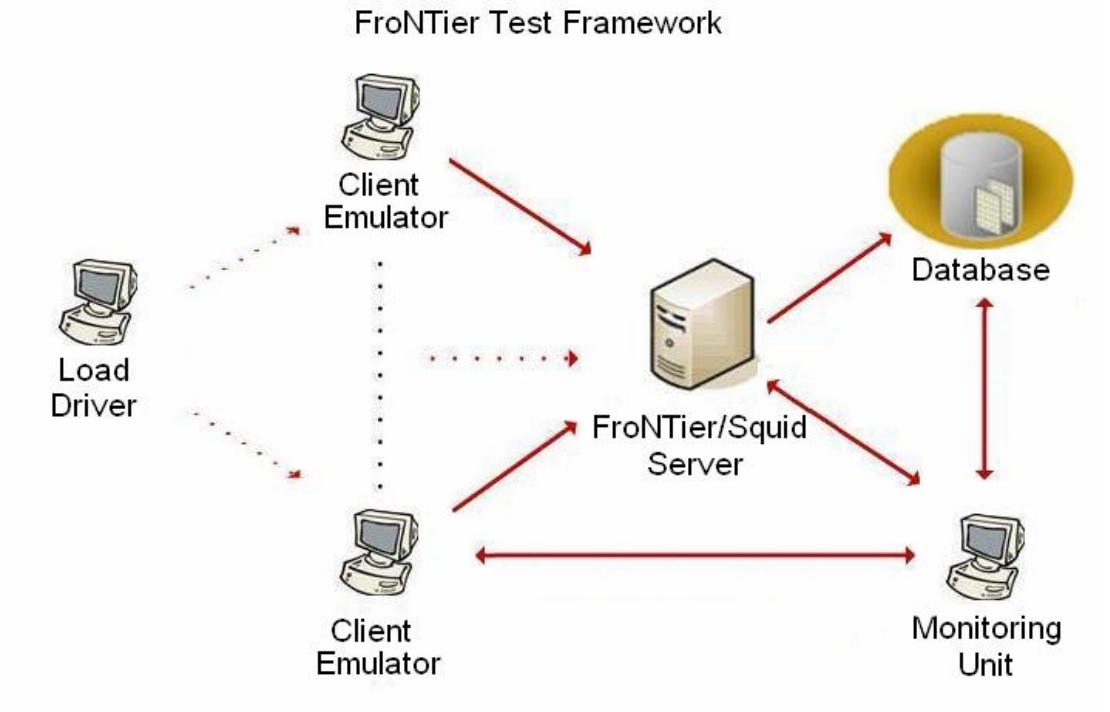


Figure 6 - FroNtier Test Framework Components

The following sections describe the control unit (load driver and integration unit) and the monitoring unit of the test framework.

4.10.1 Test Control: Load Driver and Integration Unit

The test control unit of the test framework is composed of the load driver and the integration unit.

Load drivers are tools used to load the system in order to measure its performance. Commonly, a load driver is an automated software tool that connects externally to the system under analysis, and acts as a user putting the system under a predefined workload. These tools can simulate many users in a very controlled and repeatable way.

The load driver developed to run the FroNtier tests consists of a program that starts the test clients in the client nodes, and a client program that, in each client node, reads a client script with a list of database queries and executes them against a given database.

Usually, user emulators are controlled by a script file that contains the workload the user should exercise. This script file contains the definition of the values of most of the workload parameters, such as, the characteristics of the requests/queries, the delay between requests, the client configuration, and the data access method. The client script for FroNtier tests is composed of two files: the client configuration script and the client workload script. The client configuration script is a shell script that contains the client configuration parameters as the zip level to be used by the client, the database connection string, client workload script file location, etc. Conversely, the client workload script contains the database queries to be executed by each client. The following code is an example of a client workload script (written in XML):

```
<?xml version="1.0" ?>
<queryList>
    <query>
       <select>
         <col>string</col>
       </select>
       <table>T_STRING</table>
       <where>rownum &lt;= 10000</where>
    </query>
</queryList>
```

Other system parameters are defined together through a global configuration file (additionally to the client scripts). This configuration file is also used by the integration unit of the test framework. The following code is an example of a configuration file (written in XML):

```
<?xml version="1.0"?>
<testConfig>
   <testNodes>
      <client mon="on">lxb0677</client>
      <client mon="on">lxb0678</client>
          <client mon="on">lxb0679</client>
    </testNodes>
    <monitoring>
          <server>lxb5556</server>
          <server>lxfsrk402</server>
    </monitoring>
    <execution numBoxes="17" numThreadsPerBox="3"
      numClientsPerBoxInit="1" numClientsPerStep="1" stepPeriod="1800"/>
    <locations baseDir="/afs/cern.ch/user/l/lramos/work/perf/"/>
    <clientConf testClientCmd="runFrontierClient.sh"
                testClt_queryListFileName="queryList.xml"/>
</testConfig>
```

In this file, several parameters are defined as the name of the client nodes and the servers to be monitored, the ramp up strategy (number of clients per box, number of clients to increment, and step period), test directories, test executables, and other configuration files (as the client workload script file).

In each test node, the client configuration script (shell script) is executed. It reads the client workload script (XML) and executes the Frontier client test (C++) that uses the CORAL/FroNtier Plug-in (C++) to execute the defined queries against the FroNtier server. Each client executes the queries, gathers the database results, and outputs the defined metrics (as, for example, throughput) until a shutdown message is received fom the central load driver. Meanwhile, the integration script is executed to gather the metrics printed by each test client.

Using these configuration files, the framework manages the execution of the experiments through a list of programs. The following listing shows an example execution of the program that starts the experiment:

```
$ ./startClients.py test_run_64
1 job(s) started…
ssh -x lxb0677 '…<setup_environment_commands>…
FroNtierAccess_CoralPerfClient /dbdev/Oracle queryList.xml  1 0
> $FRT_TEST_DIR/logs/test_run_64/log_lxb0677-0.log
2> $FRT_TEST_DIR/logs/test_run_64/log_lxb0677-0.err'
```

The only command line parameter is the name of the experiment that is used to identify the experiment. Additionally to this program, the framework includes programs to check the status of the execution, and to finish the execution. Examples of the execution of these programs follow:

```
$ ./statClients.py
Test clients running at lxb0677 -> 1

$ ./stopClients.py
ssh(10069) Received disconnect from 128.142.196.68:Command terminated on
signal 15.
```

The integration unit is the part of the framework that gathers client and monitoring data into one repository by parsing the test client log files and querying the monitoring unit. This ensures that the data gathering process is not synchronized with the clients' execution.

This integration unit consists of a single program that can be executed anytime provided that there are execution logs available for the experiment. The following example shows the execution of the integration unit:

```
$ ./analiClients.py test_run_64
Reading /afs/cern.ch/user/l/lramos/work/perf/logs/test_run_64
Downloading lemon plots to disk...
Registering jobs status...
Done!
Report available at logs/test_run_64/index.php
```

The result of the analysis of the client log files is a web report containing all the data gathered from the clients and monitoring unit. This web report contains the following information:

- experiment name;
- experiment configuration summary (configuration files);
- experiment execution time;
- servers and clients monitoring information;
- results (metrics).

The results section shows the following information:

- individual client results with execution time, number of queries, error reports, individual throughput, etc.;
- instant aggregated throughput (plot with the aggregated throughput every 2 minutes of the experiment);
- aggregated throughput per number of clients (plot with aggregated throughput for each ramp up stage with different number of clients);
- table with aggregated throughput values per number of clients.

Appendix I shows an example of these web reports and details the servers' monitoring plots. Although further scripts could be developed to parse each web report and gather information from each experiment in an automated manner, this step is done manually.

4.10.2 Test Monitoring

This section documents the extensive study of the state of the art of monitoring tools, more specifically, Linux performance monitoring tools. This study helped on the decision of what monitoring tools or system should be used as the monitoring unit for this evaluation.

A monitor is a tool that observes activities on a system. Usually a monitor observes a system, gathers statistics, analyze the data, and presents results. Some monitors identify problems and suggest solution.

Though monitors are not specific to performance analysis (they are used by programmers, system administrators, and system managers), monitoring is the first and key step in performance measurement.

A monitor can be event driven or timer driven. Event driven monitors are activated when an event occurs. Timer driven monitors, or sampling monitors, are activated at a fixed time intervals that can be configured. If the event are to frequent event driven monitors can introduce a big overhead. If time intervals are to big, timer driven monitors can miss important system activities or events.

Another way to classify monitors is according to the method used to display the results. Online monitors show system status continuously or at a given frequency, while batch monitors are run once and collect results that can be analyzed later.

An alternative way to monitor a system is to analyze the system logs that are normally produced. With this monitoring technique, one does not need to build a monitor, but needs to build a log analyzer instead. Moreover, logs may not contain all the information desired.

The Linux operating system has a considerable number of performance tools ([71], [72]), being most of them performance monitoring tools.

Performance is influenced by many factors, for example, processor speed, memory size, number of network or disk controllers, size and speed of disks, the workload, the data access patterns, the user usage patterns, etc.

Performance monitoring tools are typically used to identify performance bottlenecks. The typical measurement elements are CPU, memory, network, and disk I/O. Each of these elements has different performance metrics that can be monitored, for example, the CPU utilization and the load average are two metrics for CPU monitoring. Understanding how the system behaves in terms of these elements is the key to identify performance problems.

Before using a given performance tool for monitor the behavior of a system, one needs to understand how each of these tools work. Usually each tool uses a specific measurement technique for a given metric. And, although no tool displays all statistics, some of the tools

display the same statistics. These are reasons to use different tools for the same measurements to get deeper insight.

Generally, the act of observing a system modifies its behavior. Performance tools change the way the system behaves when gathering information about it. When using the tools, this performance overhead introduced by the tool itself should be taken into account. Typically, low overhead tools give a less precise view of the system; while high-overhead represent a bigger price to pay for deeper insight into the system behavior.

The kernel of the Linux operating system is primarily responsible to control how processes access the devices available on the system. A common way to optimize the performance of the system is to change these parameters. The /proc virtual directory contains a number of files describing the system's status; including running processes. Some of the files in /proc/ contain the values of kernel parameters. These files can be edited by the user. Alternatively, the /sbin/sysctl command, and the file /etc/sysctl.conf are used to view, set, and automate kernel settings in the /proc/sys directory.

The following sections describe what metrics are typically used for each system element, and what tools are used to monitor these metrics.

4.10.2.1 CPU

The CPU utilization is one of the most important metrics to monitor in a system. Tools shown in this section help answer questions like: where is the processor spending time (operating system or applications)? Is the processor idle? Are there too many processes trying to run?

The most basic tool to measure the system's CPU load in Linux (and most UNIX based operating systems) is uptime [74], which displays the load average of the system.

```
$ uptime
17:06:14  up 7 days,  5:06, 83 users,  load average: 1.24, 1.01, 1.34
```

The load average can be defined as the average number of tasks that are runnable. Runnable tasks are those that are either currently running, or those that can run but are waiting for a processor to be available. Three numbers are typically shown for the load average over the periods of the 1, 5, and 15 last minutes.

For ideal CPU utilization, the maximum load average should be equal to the number of CPUs available. This number can be obtained by looking at the contents of /proc/cpuinfo that shows a list of CPUs available, and respective characteristics. A load average less than 2.00 on a two-CPU machine indicates that the processors still have additional free cycles. The same would be true on a four-CPU machine with a load average less than 4.00, and so on.

The analysis of the load average is not as straight forward as it might seem as the formal definition of the load average is the sum of the run queue length (sampled every 5 seconds), and the number of jobs currently running on the CPUs. Thus, load average also counts as runnable all jobs in the run queue waiting on disk, or on I/O to a distributed file system. Moreover, the internal calculation of the load average is not a geometric average, but rather an exponentially-damped time-dependent average [74]. Load average is a moving average where more weight is given to the latest data (also referred as exponentially weighted moving average). This type of moving average reacts faster to recent value changes than a simple moving average.

The second most used metric for CPU is the CPU utilization. The CPU utilization represents the percentage of time that the CPU was effectively used. A value of 100% means that there are no free cycles available. This value is given for each CPU available in a given system. A CPU bottleneck is usually detected by a high CPU utilization coupled with a high load average.

In a Unix based system, the CPU can be in one of the following seven states:

- User – the CPU is executing at user level, i.e., running users' programs including library calls;
- System – the processor is executing at system level, i.e., kernel code on behalf of the application;
- Nice – the CPU is executing at user level with 'nice' priority. 'nice' is a command used to set the scheduling priority of a job (both lower and higher than the default);
- Idle - the CPU is not doing actual work, and the system has no disk or network I/O requests pending;
- iowait – the CPU is idle, and the system has a disk or network I/O request pending;
- irq - the CPU is executing high priority kernel code handling a hardware interrupt (interrupt handles have very high priority and run very quickly);
- softirq - the CPU is executing kernel code handling a hardware interrupt but with low priority, also called, soft-interrupts. Typically happens immediately after an interrupt handling.

There are several tools to analyze these states, and the overall CPU utilization.

The Multi Processor Statistics tool (mpstat, provided by the sysstat rpm) [75] reports processor related statistics.

```
$ mpstat 5 -P ALL
CPU  %user  %nice  %system  %iowait  %irq    %soft  %idle    intr/s
all  1.30   19.40    2.90    0.50    0.50    0.70   74.70   2692.60
0    1.60   16.40    1.80    1.00    0.80    1.40   77.00   2583.20
1    1.00   22.40    4.00    0.00    0.20    0.00   72.40    109.20
```

The previous example shows the output of mpstat. It shows utilization details of each of the available CPUs. mpstat makes real-time measurements of the CPU utilization, and accepts the length of the measurement (in seconds) as a parameter. In the example above, we can see the CPU utilization percentages during the 5 seconds following the execution of the command.

The first column of the output (CPU) produced by mpstat is the processor number; the value 'all' indicates that statistics are calculated as averages among all CPUs. The values shown represent, for each CPU (each line), the percentage of the total time the CPU spent in each state (column). For example, in the 5 seconds following the execution of the command, CPU 1 spent 72.4% in an idle state. The last column, 'intr/s' shows the total number of interrupts received per second by the CPUs.

As mpstat, the Virtual Memory Statistics (vmstat) [76] is a real-time performance monitoring tool. Vmstat has a broader usage than mpstat as it provides information about the run queue, the CPU usage, and the memory usage (see next section for more details on memory usage analysis). Vmstat also provides information for individual CPUs.

```
$ vmstat -m 5
Procs|        memory        |  swap  |   io    |  system  |    cpu
r  b | swpd free buff cache | si  so | bi   bo |  in   cs | us sy id wa
3  0 | 1066  21   38   509  |  0   0 | 65  118 |  21   10 | 37 10 44  9
1  0 | 1067  21   38   509  |  0   0 |  0  316 | 449  894 | 54  9 37  0
```

The previous example shows the output of vmstat. Each line of the output is a sample taken for a given period of time, in the example, 5 seconds. Each column of the output as the following meaning (sections memory, swap and io are covered in the following sections):

- procs – r - the size of the run queue is the number of active tasks waiting for CPU;
- procs – b – the number of blocked processes waiting for I/O;
- system – in – number of interrupts fired;
- system – cs – number of context switches occurred, i.e., number of changes of running process on the CPU;
- cpu – us – percentage of CPU user time (including nice time);
- cpu – sy – percentage of CPU system time (includes irq time and softirq time);
- cpu – id - percentage of CPU idle time.
- cpu – wa – percentage of CPU I/O wait time;

Vmstat offers a slight more compact version of the CPU metrics as mpstat, some insight on the run queue size adding to the load average values, and the number of context switches that can be related to the number of interrupts.

Both mpstat and vmstat are tools with low performance impact on the system which make them good candidates for constant monitoring.

The commands ps (see section 4.10.2.2), iostat (see section 4.10.2.3), top, and sar (see section 4.10.2.5) also provide CPU usage metrics like mpstat, but are generally more limited.

The /proc file system provides a way to analyze and control the Linux operating system, more precisely, its kernel. Here one can check kernel data structures, system characteristics, system parameters, etc.

The file /proc/stat contains information about kernel activity since the system first booted. Here one can get absolute numbers for the typical CPU usage metrics; like time spent executing in user mode, time spent executing in kernel mode, number of interrupts served, time spent waiting for I/O operations, number of context switches, number of processes, and threads created, etc. This file can be useful for fine grained analysis of CPU usage with the disadvantage of needing further processing and calculations.

Procinfo is a tool that analyzes the /proc filesystem, and gives the user a report similar to the one provided by vmstat. Additionally, procinfo shows the number of interrupts generated by each device, allowing the analyst to identify the exact bottleneck device, if that is the case.

The time command gives the time a command took to execute, and divides this time in real time (time from the beginning to end of the execution), user time (time that the CPU spent executing this application's code), and system time (time that the CPU spent executing system code on behalf of the application).

System level limits can sometimes be an issue, as a specific operating system limit can significantly degrade the performance of the whole system. The command ulimit provides the ability to analyze and control the resources available to user processes. Limits can be imposed on items like the maximum CPU time a process can use, the maximum size of a file,

the maximum processes for a given user, the maximum memory for a single process, etc. All these limits can be controlled with ulimit.

4.10.2.2 Memory

Before doing an overview of the tools for analyzing the system memory usage, some basic concepts about memory in the Linux operating system are introduced.

Although memory speeds continue to increase, reading and writing to memory takes longer than executing code on the CPU. CPUs spend a significant amount of time idle waiting for memory operation. This is the basic reason why memory usage is a common performance bottleneck.

Any given system as a fixed amount of physical memory (RAM). Typically, this memory is divided in fixed-sized pages. Instead of navigating in the memory byte by byte, the operation system uses these pages as the basic unit of navigation.

Virtual memory is the capability of the operating system to move less frequently used data from memory to disk while presenting the applications a virtual memory space supported both on physical memory and disk. Thus, if the physical memory of a system is not enough to run the application the hard drive is used. The disk space used of this is called the swap space. Although swapping is a very good technique to run processes that demand a lot of memory, it can be extremely slow as it resides on disk access speed, typically several orders of magnitude slower than memory access. For this reason, the swap can quickly become a bottleneck, and its management can have enormous consequences on the system performance.

Conversely, if there is too much physical memory available, the operating system will move frequently used files to the physical memory to avoid disk I/O. This can greatly improve the performance of a system if the accessed files fit into the memory cache. This is called disk cache and differs from processor cache.

Extra physical memory available is also used for buffering data that needs to be written to disk. If an application has to write some data to disk that takes a long time, the operating system can let the application continue execution by placing the data on these data buffers. With this the application does not need to wait for disk I/O, and the buffers can be written to disk at some point in the future.

Linux tries to use as much memory as possible for the cache, and buffers so a system memory can be almost full only with these. This is not necessary a bad thing, as the operating system does a dynamic management of the memory. For example, if the system needs the memory used for cache for something considered more important, the cache is flushed to disk and that memory space can be used. Future access to files in that cache fragment has to depend on disk I/O.

The Virtual Memory Statistics (vmstat) command (see previous section) is primarily a tool for system memory usage analysis. From the output of the command, the sections memory and swap refer to memory analyzes, and show the following information:

- memory – swpd – the amount of memory currently swapped to disk (in kilobytes);
- memory – free – the amount of memory not being used (in kilobytes);
- memory – buff – the size of the system buffer for I/O data structures (in kilobytes), memory space used to cache data structures before writing them to disk;

- memory – cache – the size of the system disk cache (in kilobytes), memory space used to store data previously read from disk;
- swap – so – the amount of memory swapped out to disk (in kilobytes); reflects the swapping activity as data is swapped out to the swap space, it is a normal part of the memory management procedure as unused memory is moved to disk;
- swap – si – the amount of memory swapped in from disk during the sample (in kilobytes), represents pages that are swapped back to the physical memory. Unlike swap out operations, swap in operations denote that swapped memory pages are, in fact, being needed; usually shows that the total amount of memory available is not enough for optimal execution;

The columns 'swpd', 'si', and 'so', are useful to check whether the system is swapping, and if so, swapping rates can be registered.

The command ps (Process Status) [77] is probably one of the oldest and feature-rich commands to extract performance information. It is especially useful for analysis of process specific performance statistics. Ps provides static information about running processes (such as the command name), and dynamic information (such as the memory and CPU usage). The command 'ps aux' shows the total percentage of system memory and the amount of physical memory that each process consumes. Ps shows the virtual memory size that a given process is using, the amount of the process' physical memory, the amount of memory that the executable uses, etc.

Under the /proc file system one can find the files /proc/meminfo and /proc/slabinfo that capture the state of the virtual memory. On these files we can find the total amount of physical memory of the system, the total amount of unused memory, the size of the buffer cache for I/O operations, the amount of cache memory that has been swapped out in the swap space, the size of the swap space on disk, the amount of memory used by the kernel data structures (/proc/slabinfo), etc. These files can be captured periodically to establish a pattern of memory utilization.

Also under the /proc file system, the /proc/<pid> virtual directory (where pid is the PID of the process) contains information about the process that is not available through any other performance tool. For example, /proc/<pid>/status is a file that contains most of the information shown by ps, plus some more fields like the amount of memory the libraries are using, the size of the process' data, or the amount of virtual memory locked by the process. /proc/<pid>/maps contains details about how memory is allocated for a given process, such as files used by the process and their location.

The virtual directory /proc/sys/vm/ contains a list of files where kernel memory settings can be seen and changed. Examples of these are the total buffer memory size, and the thresholds for swap management.

The file /proc/modules contains the list of modules loaded into memory. It can be useful to detect modules that are not used and fill the memory space unnecessary.

Many other commands exist for system memory analysis. For example, the tool swapon can be used to list active swap devices; the tool free gives an overall view of how the system is using the available memory; slabtop can be used to analyze the internal kernel caches sizes;

ipcs is used to track what processes are allocating system shared memory, semaphores, or memory queues; etc.

4.10.2.3 Disk I/O

Although the speeds of the I/O devices continue to increase, I/O throughput and latency are still orders of magnitude slower than equivalent memory access. Because many workloads have a substantial I/O component, I/O can easily become a significant bottleneck to overall throughput and application response times.

The I/O access patterns exercised by the applications are an important characteristic of the workload of a given system. Disk drives are able to handle large sequential transfers better than small random transfers. Applications typically rely on the capability to access data in random locations on disk. As a result, the I/O access patterns tend to be a mix of sequential and random accesses. The I/O system performance can be severely impacted by the access pattern of the workloads.

When analyzing I/O performance, it is essential to know the limitations of all the components of the underlying storage system. The analyst needs to understand not only the storage devices limitations, but also the way the system is interconnected and structured. One should have in mind that the I/O performance cannot exceed the performance of the underlying hardware. In addition, it is not obvious to distinguish between hardware and software bottlenecks.

The main objective of the analyst in terms of I/O operations is to increase data transfer rates to and from storage devices and reduce I/O latencies. For this reason, performance is often evaluated in terms of overall throughput and latency of I/O requests. The latency of a request is the time the request had to wait for the I/O operation.

If software optimization is not enough, many hardware solutions for increasing I/O performance exist including faster disk drives, better disk type like SCSI disks, and larger disk cache sizes. Another hardware solution to increase the I/O performance of a system are the RAID systems (Redundant Array of Independent Disks) that increase access parallelism by striping data across multiple disk drives. RAID systems offer better I/O performance by using multiple hard drives to share or replicate data among the drives. RAID systems combine multi low cost drives into a single logical unit that is seen by the operating system.

The disk I/O queue is where the kernel puts the I/O requests. Here the requests can be grouped before the actual I/O operations.

The Linux operating system offers several different tools for performance evaluation of the disk I/O subsystem. Typical monitored metrics are: the total number of I/O operations processed by the system, the number of I/O operations per logical disk drive, the overall I/O transfer rate. These tools can be used to identify I/O bottlenecks, what disks are being used, how much I/O each disk is performing, and what the latencies are.

As for CPU and memory, vmstat (see previous sections) also produces statistics about the usage of each disk in the system, within a given time interval. The statistics collected by vmstat are: the total number of reads and writes, the total number of sectors read and written, the amount of time spent reading and writing to disk, and the total amount of time spent waiting for I/O to complete. Vmstat offers a rather basic but useful set of disk usage metrics.

The iostat command [78] is a disk I/O specific tool that monitors system I/O activities and generates reports. As vmstat and mpstat, iostat analyzes the system during a given interval of time. These reports can be used to change system I/O configuration and, for example, balance the I/O load among physical disks. An example execution of iostat is shown below:

```
$ iostat -k 5
Linux 2.4.21-47.0.1.EL.cernsmp (lxplus055.cern.ch)      12/19/2006

Device:    tps    kB_read/s    kB_wrtn/s    kB_read    kB_wrtn
hda       37.60      209.60       186.40       1048        932
hda1      10.00      206.40        13.60       1032         68
hda3      24.00        3.20       153.60         16        768
```

Each line reports statistics about one logical disk. The disk/device name, in the first column, is followed by the following values:

- tps - number of I/O requests that were issued to the logical disk;
- kB_read/s and kB_wrtn/s –data rate read from and written to the logical drive;
- kB_read and kB_wrtn - amount of data read from and written to the logical drive;

With the option –k, one can see the same numbers in number of blocks instead of Kbytes. The option –x shows an extended I/O usage report with information such as the average wait time, the average service time, the average request size, average size of the request queue. Iostat also provides CPU utilization that can sometimes be useful in comparing directly against the I/O activities.

The tool lsof (list open files) shows the user which processes have a particular file open. It can also show what processes are using files in a particular directory. In the Linux operation system, it is not easy to find out what process is causing a given I/O activity. Although lsof doesn't show the amount of I/O on each file, it can show what processes are generating I/O operations.

/proc/sys/fs/* contains a number of files to control kernel parameters related to the file system. For example, /proc/sys/fs/file-max contains the maximum number of file handles a user can get. /proc/partitions contains a list of all disks and partitions available with respective name and size.

The df [79] command reports file system disk space usage, and how each partition on the disks are mounted to the file system directories.

Hdparm is a powerful tool that measures hard disks speed, and tune how the kernel is using the hard disks available on the system. This tool includes a benchmark for disks, and can greatly help improving the I/O operations.

4.10.2.4 Network I/O

Computer networking and network performance analysis [80] are computer disciplines by themselves. The Linux operating system supports features such as packet forwarding, firewall

operations, proxy, tunneling, and aliasing; and implements the three lower network layers: link, network and transport.

At the link level (e.g. Ethernet), the system sends information to the network as a series of frames. When analyzing a system, it is fundamental to know the speed of the underlying physical network to which the system is connected. Current Ethernet interfaces typically support 100Mbps and 1Gpbs. The underlying network elements like switches and routers can be the bottlenecks as they may or may not support such speeds.

At the transport layer (typically TCP or UDP), Linux uses the socket/port concepts: local applications use network sockets to connect to ports on remote systems. Tools exist to monitor the traffic on a given network port.

Upper network layers may send information units much bigger than the size of a transfer unit of a given layer. Each layer breaks everything up into transfer units to send them to the network. At the link layer, the maximum size of data in a frame is called the MTU (maximum transfer unit) that influences the datagram size at IP level. At TCP level, the MSS (maximum segment size) is the maximum size of data in a packet and is set in each host. Nevertheless, the TCP packet sizes may change depending on which hosts are communicating.

Generally, bigger sizes offer better performance (smaller header overhead), but increase the probability of frame loss. The value of this configuration parameter should depend on the quality of the network to which the node is connected.

In terms of network monitoring, the common performance metric is the rate at which data traffic is flowing through each of the network layers.

The Linux operating system provides tools for network administration, monitoring, troubleshooting, and security. This section explores some of these tools. Different monitoring tools typically work with different network layers.

The command ping verifies if a host has a working network connection. ping uses the Internet Control Message Protocol (ICMP) by sending a small packet through the network to a given IP address. If a reply is received, the computer network connection is alive. Ping is also useful to tell how many routers exist between the source and the destination host.

ifconfig (Interface Configure) displays information about the network interfaces like interface address, the MTU, and counts of packets received and sent (successful and erroneous). It also shows basic network statistics since the system as booted that, if monitored, can be of great interest.

traceroute tracks the network path between two hosts registering: the routers of that path, and the time spent between each router.

The route command can be used to manage the route tables of a system.

mii-tool is an Ethernet specific tool primarily used to configure Ethernet devices; it also shows information about the device, like link speed and duplex settings.

ethtool is similar to mii-tool as it is used to manage the Ethernet devices of the system; but it is more powerful than mii-tool with more options and statistics.

Ethereal and tcpdump are very useful tools for network traffic monitoring and analysis as they capture all the packets going through the wire.

host and nslookup use a Domain Name Server (DNS) to translate a host name to a IP address or vice-versa. nslookup is quite more limited than host.

The netstat utility [81] is one of the most frequently used tools for monitoring network connections and collect network statistics in a Linux system. netstat displays a large amount of information related to the networking subsystem like the list of active sockets for each network protocol, the network routes, and statistics of the network interfaces (number of incoming and outgoing packets, and the number of packet collisions).

The basic feature of netstat is to display the list of existing sockets. The example below depicts a basic netstat execution example.

```
$ netstat
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q  Local Address          Foreign Address        State
Tcp        0  56236  lxplus009.cern.ch:ssh  abpc10881.cern.ch:1752 ESTAB
tcp        0      0  lxplus009.cern.ch:5206 pclhcb49.cern.ch:x11   ESTAB
tcp        0      0  lxplus009.cern.ch:3472 lxmrra3801.cern.ch:ssh ESTAB
```

The first column contains the protocol of the socket. The second and third columns contain the number of bytes currently in the socket queues. Following columns show addresses and port information. The last column shows the socket state.

Netstat can also show the amount of network traffic flowing throw the network.

```
$ netstat -i
Kernel Interface table
Iface MTU Met RX-OK RX-ERR RX-DRP RX-OVR TX-OK TX-ERR TX-DRP TX-OVR Flg
eth0 1500   0 447134403  0      0    0  454272843  0      0      0 BMRU
lo  16436   0 34022740   0      0    0   34022740  0      0      0 LRU
```

The first column showing the interface name is followed by the MTU column that shows the current MTU for that interface. The RX and TX columns show how many packets were received or transmitted error-free (RX-OK/TX-OK), damaged (RX-ERR/TX-ERR), dropped (RX-DRP/TX-DRP), and lost (RX-OVR/TX-OVR). The last column shows the configuration flags of the interface. These statistics on the number of packets sent and received is of high importance to investigate network performance.

Netstat can display the kernel interface-specific counters for statistics about UDP/TCP traffic that the system has sent and received since the last boot. These counters include the Simple Network Management Protocol (SNMP) Management Information Base (MIB) and other Linux-specific counters. The list of counters is available in the files /proc/net/snmp and /proc/net/netstat files. The following example shows these network counters for UDP traffic.

```
$ netstat -s -u
Udp:
    18219472 packets received
    67 packets to unknown port received.
    5624 packet receive errors
    27962977 packets sent
```

Additionally, netstat can map network sockets to the processes using them; and, as ifconfig, it can display information about the network interfaces of the system.

The tool Gkrellm provides a graphical view of some performance metrics. Its unique feature is that it shows which system services are consuming different amounts of network bandwidth.

Lsof (see section 4.10.2.3) provides a list of currently opened files that includes network files, i.e., network sockets. This tool can be used to list opened TCP and UDP sockets and associated processes.

Several Linux kernel parameters can be used to control and improve the network behavior of a system. These parameters can be found under the file system /proc/sys/net. For example, /proc/sys/net/core/wmem_default and /proc/sys/net/core/rmem_default contain the default sizes of the TCP send and receive windows, and /proc/sys/net/core/wmem_max and /proc/sys/net/core/rmem_max contain the maximum sizes of the same windows. These four parameters can greatly influence the network performance of the system. /proc/sys/net/ipv4/tcp_timestamps controls the use of timestamps in the TCP header, and /proc/sys/net/ipv4/tcp_sack enables TCP selective acknowledgements.

The Andrew File System (AFS) and the Network File System (NFS) are distributed network file systems that enable transparent access to remote disk resources. This is done by incorporating a file system from a remote machine into the local file system. The nfsstat tool can be used to collect NFS statistics as, for example, the number of NFS API calls (reads and writes). These numbers can be helpful to understand the performance of a system that uses these technologies.

Several tools have been presented in this section: tools to inspect connections on the system (mii-tool and ethtool), tools to monitor the amount and type of packets flowing through the network interfaces (ifconfig, gkrellm, netstat), tools that display the different types of IP traffic and the amounts of each type of traffic (gkrellm, netstat), tools for system administration (ifconfig, host, nslookup, and the kernel parameters), tools for network troubleshooting (ping, route, traceroute), and tools for network traffic content analysis (ethereal and tcpdump). All these tools can, in some way, be used for analyze and optimize the performance of a system.

4.10.2.5 General Monitoring Tools

top is a very complete performance monitoring tool for Linux as it sums information seen previously in other tools like ps, vmstat, uptime, etc. top provides a real-time interactive overview of the system load (CPU and memory), and all running processes that are shown in a sortable list.

The SAR utility is another rather complete and versatile performance tool that collects a wide range of system activity metrics such as the CPU utilization, the rate of context switches and interrupts, the rate of paging in and paging out, the shared memory usage, the buffer usage, and network usage. Sar distinguishes itself from other performance monitoring tools because it constantly collects and automatically logs the performance metrics in a set of log

files. Moreover, the format of these logs and, as vmstat, the number and frequency at which the tool collects statistics can be configured. These features together greatly improve the easiness of a performance evaluation.

The Nmon tool, developed by IBM, is designed for monitoring and analyzing performance data in AIX and Linux systems. The long feature list includes monitoring of CPU utilization, memory usage, Kernel statistics, run queue information, disk I/O rates, disk I/O transfers, disk I/O read/write ratios, file systems space, network I/O rates, network transfers, network read/write ratios, paging space, paging rates, Network File System (NFS), and several AIX specific features. Nmon also includes a tool to generate graphs from its output.

Although, these three generic tools are very complete, and include enough features to analyze the performance of a system, generally, specific tools offer more details: for example, mpstat for CPU, vmstat for memory, iostat for disk I/O, and netstat for network.

4.10.2.6 Tracing and profiling tools

The previous sections described mostly system-wide performance tools. Tools with process specific analysis capabilities such as ps or top were referred. All these tools help to determine how the system, or a specific process, is behaving in terms of CPU utilization, memory, etc. In this section two other techniques for process specific performance analysis are described: tracing tools and profiling tools.

Tracing tools can help determine what libraries and system calls in a given process are being made and how long those calls take.

The tool strace (System Trace) [82] traces the system calls that a program makes while executing. System calls are function calls made into the Linux kernel by or on behalf of an application. strace can show the system calls made by the application and by the libraries the application uses. It can provide a table showing the frequency of each system call and the total time spent in calls of that type. This tool helps understanding how a given application interacts with the kernel.

ltrace (Library Trace) is similar to strace, but it lists calls made by an application to libraries. Ltrace lists library calls and shows the name of the functions, arguments, return values, and the time spent in each call. It helps identifying what library calls are being made and how long do they take.

Ltrace lists only the name of the library functions. In order to identify the exact library where the listed functions are, one can use tools like ldd or objdump.

Profilers are another way of analyzing the behavior of an application. Profilers help tracing the execution of a program and time the execution of the different parts of the program, typically, each function of the program. This information can be used to tune the application by identifying the most frequent and time consuming parts of the code. Three profilers are described here: gprof, that analyzes the call graph of a given process; oprofile, a more complete and system-wide profiler; and memprof, a memory profiler.

gprof is a profiling tool that displays the program's call graph. Given a program, gprof lists the called functions of the program and calculates the amount of time spent in each call. gprof instruments the source code of the application under study and then runs the application to produce a sample file. To accomplish this task, the application source code must be available for the instrumentation. Although gprof shows the exact list of functions being

called, and make an approximation of the time spent in each call, it will most likely add some overhead and change the performance behavior of the application being measured.

Oprofile is a system-wide profiler that tracks where CPU time is being spent. It can be used to analyze single processes or an entire system. oprofile is a lower-overhead tool than gprof because it does not require code instrumentation. It measures some events not supported by gprof, but does not build call graphs by default like gprof. Using processor counters, oprofile can measure very low level information like cache misses, floating point operations, etc. Instead of recording all the processor events, it uses sampling which only introduces a small overhead, but makes the analysis of results complex. The oprofile reporting tool displays the collected samples and which executables or libraries are responsible for them. Oprofile annotation tool can extract information about performance samples and, for example, link samples to specific source code lines. Oprofile also offers a GUI that enables the graphical control of the tool.

Gprof and oprofile only work with static languages like C or C++. For dynamic languages like Java, python or Perl other profiling tools exist with similar functionalities.

Memprof is a profiler of memory usage that shows the amount of memory a program is using and which functions are responsible for that. Memprof updates information dynamically as the application is running.

Valgrind is a tool for memory debugging, memory leak detection, and profiling. It is a high overhead tool but it can be very useful detecting problems like usage of uninitialized memory, access to already freed memory, accesses to non allocated memory, and memory leaks.

ld.so is the Linux loader that loads the libraries of dynamically linked applications (application that use shared libraries). It connects symbols that the application uses with the functions provided by the libraries. As different libraries are originally linked at different and sometimes overlapping places in memory, the linker needs to check all the symbols and manage all the needed memory relocations. The Linux loader runs automatically before the execution of any dynamically linked application. Although this step is done transparently for the user, it can significantly slow down the startup up of an application. The tool prelink can be used to speed up the linkage process. It reorganizes the libraries of the entire system so that they do not overlap in memory. If configured, the Linux loader command (ld) can print statistics about the linkage process. As this can be an application bottleneck, these values can be rather useful.

4.10.2.7 Distributed Monitoring with LEMON

Monitoring a distributed system is more difficult than monitoring a centralized system. In distributed systems, monitors are commonly divided in components to allow distribution. An example of architecture of a distributed monitor is a layered structure with components as an observer, a collector, an analyzer, and a presenter.

The LHC Era Monitoring (LEMON) [52] is a distributed monitoring system that uses this architecture. On every node monitored by LEMON, a monitoring agent (collector) launches and communicates with sensors (observers) which are responsible for retrieving monitoring information. The extracted samples are sent to a central measurement repository where they

are analyzed (analyzer). The user can access monitoring information in several ways (presenters) including a web interface and a programmatic API to the repository.

4.10.2.8 Monitoring Unit

In this section we have presented the performance tools that could be, in some way, useful for this thesis work. Several other performance tools exist that could complement the ones described here. A general conclusion is that there is no single tool that provides all relevant performance statistics on Linux. Almost all the tools shown in this section offer some sort of information not available on order tools.

The monitoring unit of the FroNtier test framework makes use of several of the performance tools described in this section. Some tools are directly used by the test clients for self monitoring, while others are used indirectly through LEMON.

In the client side, mpstat, uptime, and vmstat are used to collect, respectively, the CPU utilization, the load average, and the memory consumption. LEMON is used to analyze all the utilization metrics of the servers: FroNtier, Squid, and database.

4.11 Summary

This chapter presented the methodology used in the performance evaluation of FroNtier. Additionally, section 4.10 presented the test framework developed to execute the tests while section 4.10.2 documented an extensive study of the state of the art of monitoring tools.

Chapter 5 - Performance Analysis of FroNtier

The previous chapter described in detail the performance analysis methodology of the database caching system FroNtier/Squid and the test framework used to automate those tests. This chapter presents the experiments conducted based on that methodology and using that framework.

This chapter is structured in two sections. Section 1 describes the test setup in terms of hardware and software configuration of the different components of the system: FroNtier servers, database servers, and client servers. Section 2 documents the experiments detailed design, execution, and individual experiment result analysis.

5.1 Test Setup

A FroNtier setup is based on a collection of nodes, each running a FroNtier server and a Squid server that connect to a backend database server. A test system was setup parallel to the LCG 3D FroNtier production setup (see section 3.2.1). This section describes the software and hardware setup of the test clients and servers of this test setup.

FroNtier/Squid Server Setup

The FroNtier/Squid test nodes are servers with Dual Intel(R) Xeon(TM) CPUs running at 2.80GHz, 2GB RAM, 150GB hard drive, and Fast Ethernet (100Mbps) connections. Gigabit Ethernet connections were considered (for both production and test setup) but no justification for those speeds was found.

The FroNtier/Squid test nodes run on Scientific Linux 3 (a Linux distribution based on Red Hat 3 with Linux Kernel version 2.4) with Java 1.4.2_06-b03, Tomcat 5.5.0.28-11, FroNtier Servlet 3.4, and FroNtier Squid 1.0rc4 (includes Squid v2.5.STABLE7 and additional scripts). Both FroNtier and Squid are deployed as RPMs using Quattor [51].

The most important FroNtier servlet configuration options (taken from the configuration of the JDBC connection to the database) are:

- `maxActive – 10`
    - defines the maximum active database connections;
- `validationQuery - select 1 from dual`
    - defines the validation query to verify the presence of the database;
- `session-timeout - 30`
    - database session timeout.

Although there are three nodes available for the evaluation, the typical value for the number of FroNtier and Squid servers is 1.

The most important Squid cache server configuration options are:

- Size of the in memory cache: 512MB (25% of the 2GB RAM);
- Maximum Object Size: 256MB;
- Maximum Object Size In Memory: 1MB;
- Cache size in disk: 90GB;
- Squid always runs in http accelerator mode (as a reverse proxy server).

Database Server Setup

The test setup uses an Oracle database for all the tests as this is the database technology mainly used in the project. The backend Oracle Database used for the tests runs on a Dual Intel(R) Xeon(TM) CPUs running at 2.80GHz, 2GB RAM, 150GB hard drive, and Fast Ethernet (100Mbps) connections. The Oracle Database version is the 10gR2 (10.2.0.3) running on Linux Red Hat 3.

Clients Setup

The test clients run in dedicated nodes with dual Pentium III 1GHz, 500MB RAM, Hard drive with 16GB and Fast Ethernet (100Mbps). These nodes run Scientific Linux 3 (a Linux distribution based on Red Hat 3 with Linux Kernel 2.4). Clients run CORAL version 1.6.3 with the FroNtier Client plug-in version 2.5.1 (patched to output the test metrics).

5.2 Test Plan: the Experiments

After having defined, in the previous chapter, the experiment design, the workloads, the test parameters, and factors, in this section we put all these values together by defining the actual tests cases. The main choices here are what factors should be crossed, the experiment designs, and the workloads.

As seen in section 4.6, the parameters chosen as primary factors are: number of clients with levels 1, 2, 5, 10, 20, 50, 80, and 100; query size with levels 1kB, 10kB, 100kB, 1MB, and 10MB; FroNtier documents compression factor with levels 0, 1, 5, and 9; and data access method with levels direct FroNtier access, Squid cache access, access to Squid forcing cache refresh on each query, and direct Oracle database access. Additionally, the secondary factors are: number of queried tables with levels are 1, 2, and 5; FroNtier server version with levels 3.1 and 3.3; client nodes location with levels CERN tier-0 and FNAL tier-1; CORAL client connection mode with levels connect on every query and connect only once; number of FroNtier/Squid severs with levels 1 and 2; maximum active database connections on the FroNtier servlet with levels 5, 10, and 20; number of rows in each queried table with levels same as number of result rows and 10 times that number; C++ data types with levels int (NUMBER(10)), float (BINARY_FLOAT), double (BINARY_DOUBLE), and std::string(x) (VARCHAR2(x)); query data with levels random data and simulation data; query frequency with levels 0, 10, 30, and 60; and squid configuration such as the cache size, cache peering, and cache hierarchy.

As seen in section 4.8, the possible workloads are the CORAL workload, the COOL workload, and the Athena workload.

As seen in section 4.9, the possible experiment designs are the simple design, the full factorial design (including the 2k design), and the fractional factorial design.

The client ramp up strategy is defined in the global configuration file (see section 4.10.1). In every experiment, the number of client nodes, the initial and total number of client processes to run in each node, the experiment duration, and the client step number are defined. For example, if the number of clients of an experiment is 4, 8, 12, 16; the number of nodes can be 4, the initial number of processes 1, the total number of processes 4, and the client step 1. So, the test starts with 1 process running in each of the 4 nodes; in intervals of the experiment duration, the number of clients in each node will be added in 1 (the client step number) until the total number of processes (4 in this case) is reached.

The test duration of an experiment should be enough that the difference between "one successful query" and "zero successful queries" is ignorable. For example, on a test with queries of 1MB, after 30 minutes of execution on the client, if no payload is retrieved, it means that not even 0,5kBps (1MB divided by 1800 seconds) were achieved by this single client. Note that, if it is a test with 100 clients, this value is not ignorable as the aggregate throughput could be 50kBps (0,5kBps times 100 clients). So, the test duration in this case should be bigger than 30 minutes.

The metrics selected for the evaluation and gathered by the integration unit of the test framework are: the global throughput, the variability of the individual and global throughput, the stretch factor, the resource utilization, and the reliability.

The following sub-sections describe the experiments executed. Each experiment is a selection of one workload, one experiment design, and a collection of factors and levels.

5.2.1 Analysis methodology

The coherency of the results is tested by comparing the replication values against each other and identifying major discrepancies (using a verification script).

The first analysis step is to manually analyze the monitoring plots of the FroNtier servers, oracle servers, and clients, in terms of resource usage (CPU, network, memory, disk, etc.). This step enables the identification of bottlenecks: very high resource usage nearly the maximum available defines a bottleneck (100% CPU consumption, 100Mbps network utilization, etc.). Some resources as disk and memory are not that relevant in these tests as they are normally not a bottleneck. Nevertheless, it is useful to verify its utilization.

Finally, the most significant metrics, such as individual throughput and stretch factor, are selected for analysis according to the results obtained.

A word should be said about statistical analysis of the results. Some statistical techniques were used to summarize measured data, especially in representing results variability (values as geometrical mean, variance and standard deviation). Additionally, more advance statistical techniques could be used, for example, confidence intervals or regression modeling. These techniques would allow to best describe the data taken in the experiments, to estimate the contribution of each factor to the performance, and to isolate measurement errors. Nevertheless, these techniques were not used as the level of precision and time scale for that was not the desired, i.e., a faster and less statistically precise analysis was needed.

5.2.2 Experiment 1 - Access Methods Comparison

Description: compare the different access methods with a single client node
Parameters:
- Compression factor (zip level) – 5

Workload: CORAL
Factors:
- Number of clients with levels 1, 3, 5, 7 (single client node)
- Query size with levels 10kB, 100kB, 1MB
- Data access method with levels direct FroNtier access (FroNtier), Squid cache access (Squid), and direct database access (Oracle)

Experiment Design: Full factorial design with 2 repetitions
Experiment duration: 300 seconds
Number of experiments: 4 x 3 x 3 x 2 repetitions = 72

Experiment 1 - Conclusions

FroNtier Access

For direct FroNtier access, with up to 7 clients running in a single client node, the bottleneck is the CPU consumption on the FroNtier server. The only exception happens when the query size is very small (10kB) and the bottleneck is shared with the DB CPU consumption. FroNtier server CPU consumption with a single client node reaches values of 100% when 7 clients are running.

The bigger the query size, the bigger the load on the FroNtier server. This is explained by the compression work on the FroNtier server. The higher the load on the FroNtier server, the lower the load on the DB, as the bottleneck on the FroNtier server reduces the DB request rate.

Oracle Access

In this case, the FroNtier server is not used as the clients access the database directly. The DB network consumption gets to 3MBps and stops. As the network connection is 100Mbps (12,5MBps), this means that, in this case, the network is not a bottleneck.

The load on the DB CPU is never very high with 10-40% CPU consumption and a load between 0,5 and 1.

The client CPU utilization is between 10-70% and the load between 0,5 and 2,5.

None of these values represent a bottleneck but all values are quite high, particularly on the DB CPU (the CPU load gets to 1). We can consider that no bottleneck was reached and, as it will be seen in the next point, the system may well get to higher throughputs.

Squid Access

In this access mode, even using only one client node, the network utilization on the FroNtier/Squid server is very high (6MBps). The network bottleneck is assumed to be over 12,5MBps (100Mbps). On the other hand, the CPU load on the FroNtier server and on the DB is very low as all the accesses are issued to the Squid cache where the objects are cached and already compressed.

In this test case, the bottleneck is detected at the client node level where the CPU utilization goes from 50% to 100% (with 5 and 7 clients), and the load average gets to a maximum of 7 (proportional to the number of clients). These high loads on the client CPU are caused by the decompression work that as to be done by the clients, even if the query is cached on the Squid.

Throughput Analysis

The following plot shows the achieved throughput using the three access methods: direct FroNtier access (FroNtier), Squid cache access (Squid), and direct database access (Oracle). The horizontal axis represents the number of clients (#clients) and the vertical axis the throughput obtained in database rows per second (each row has 100 bytes). The query size is 1MB.

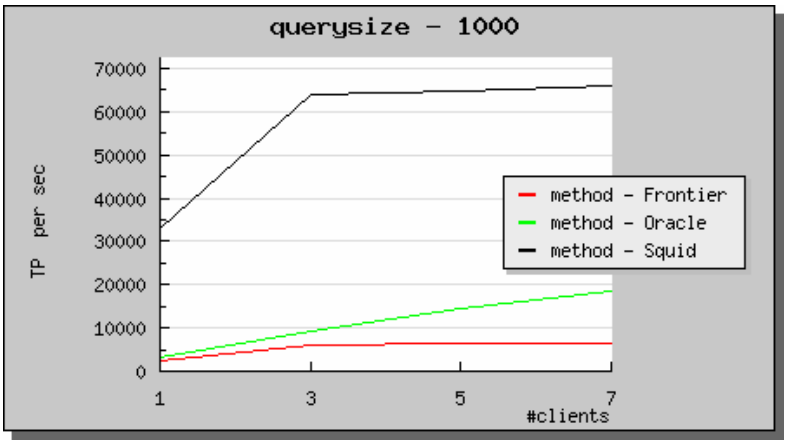


Figure 7 – Access methods comparison with 1MB queries and a single client node (throughput / number of clients)

In this plot we can see that, for 1MB queries, the maximum throughput achieved with direct FroNtier access is 6600 rows per second (640kBps). As seen in the previous point, the bottleneck here is the CPU consumption on the FroNtier server.

Oracle access maximum throughput is 18500 rows per second (1800kBps). As the plot shows, the throughput (green line on the plot) does not stabilize and may well be bigger for a bigger number of clients. This shows that the system did not saturate.

Squid maximum throughput is 65000 rows per second (6400kBps). As seen before the bottleneck here is the client node CPU utilization.

For smaller query sizes, all methods perform worse. This is explained by the overheads introduced in each result set. This performance degradation is proportional between access methods except for Oracle access that performs proportionally better than the other access methods (half the throughput of Squid) with very small queries (10kB). This is explained by the bigger overhead that FroNtier and Squid introduce in the result sets.

For all query sizes, both Squid access and FroNtier access reach the maximum throughput with only 3 clients while Oracle access scales better until 7 (or more) clients. The Squid performance bottleneck is explained by the client CPU saturation and the FroNtier access by the FroNtier server CPU saturation. Oracle access seems to stand more clients. The setup used in the next experiment will try to avoid these bottlenecks and will allow different bottlenecks to arise.

5.2.3 Experiment 1.1 - Access Methods Comparison with 4 client nodes

Description: compare different access methods with several client nodes

Parameters:

- Compression factor (zip level) – 5

Workload: CORAL

Factors:

- Number of clients with levels 4, 8, 12, 16 (4 client nodes)
- Query size with levels 10kB, 100kB, 1MB
- Data access method with levels direct FroNtier access (FroNtier), Squid cache access (Squid), and direct database access (Oracle)

Experiment Design: Full factorial design with 2 repetitions

Experiment duration: 300 seconds
Number of experiments: 4 x 3 x 3 x 2 repetitions = 72

Experiment 1.1 – Conclusions

For FroNtier access, the conclusions are exactly the same as for the single client node test (experiment 1). The FroNtier server is always saturated with 100% CPU utilization and load averages that grow up to 10 (for a dual processor machine). No stress is involved in the client side and the CPU utilization on the DB server is about 20%, even with 16 clients.

When Oracle direct access is used, the FroNtier server is not used. Here we observe the same behavior for all query sizes: very low load on the clients (10% CPU consumption with 0,5 load average), very high load on the DB server with CPU consumption of 20-90% and a load average of up to 6 (both proportional to the number of clients), and very high network traffic with 6MBps.

For Squid access, the bottleneck with several client nodes is the network with 12MBps (96Mbps) being reached with 16 clients. For query sizes of 100kB and 1MB the client CPU utilization is 50% with loads of 1,5; the CPU consumption on the FroNtier/Squid server is only 10% (servicing cached queries); and there is almost no activity on the DB server as all user queries are cached. For small query sizes (10kB), although the network throughput is very high (8MBps), the bottleneck on the client nodes is still noticed with 90% CPU consumption (load average of 3 with 16 clients). This is due to the overhead introduced by many small queries. Changing from single client node to several client nodes made the bottleneck move from the client CPU utilization to the network utilization.

The following plot shows the achieved throughput using the three access methods: direct FroNtier access (FroNtier), Squid cache access (Squid), and direct database access (Oracle). The horizontal axis represents the number of client scripts running on 4 test nodes (#clients) and the vertical axis the throughput obtained. The query size is 1MB.

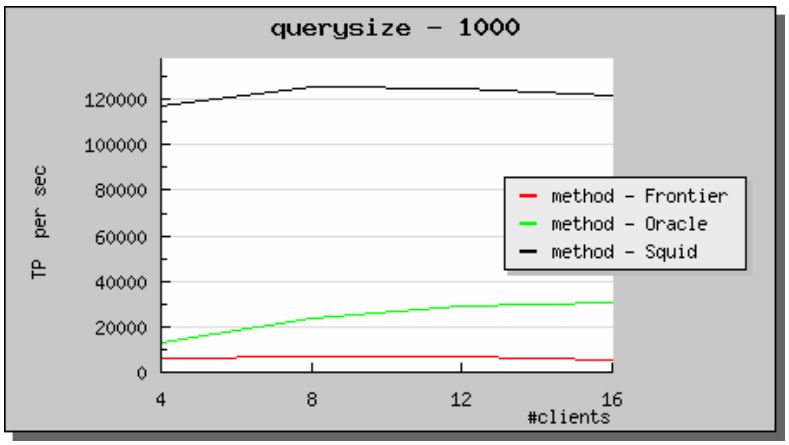


Figure 8 – Access methods comparison with 1MB queries and 4 client nodes (throughput / number of clients)

Comparing this plot with Figure 7 (experiment 1), we can identify different interesting points:

- FroNtier access shows the same behavior with saturated FroNtier server CPU;
- Squid access is stabilizing much higher with network saturation (100Mbps) instead of client CPU saturation;

- Oracle access is more stable and does not scale as good as with one client node (DB CPU bottleneck seems to be achieved);

Again in all cases, small query size of 10kB has strong impact on the performance as overhead sizes and payload sizes are of comparable orders of magnitude.

5.2.4 Experiment 1.2 - Access Methods Comparison with 5 client nodes

Description: same experiment as experiment 1.1 but with higher number of clients and fixed query size of 1MB.

Parameters:

- Query size - 1MB

Workload: CORAL

Factors:

- Number of clients with levels 10, 15, 20, 25, 30 (5 client nodes)
- Data access method with levels direct FroNtier access (FroNtier), Squid cache access (Squid), and direct database access (Oracle)

Experiment Design: Full factorial design with 2 repetitions

Experiment duration: 300 seconds

Number of experiments: 5 x 3 x 2 repetitions = 30

Experiment 1.2 – Conclusions

The following plot shows the throughputs for different number of clients and different access methods.

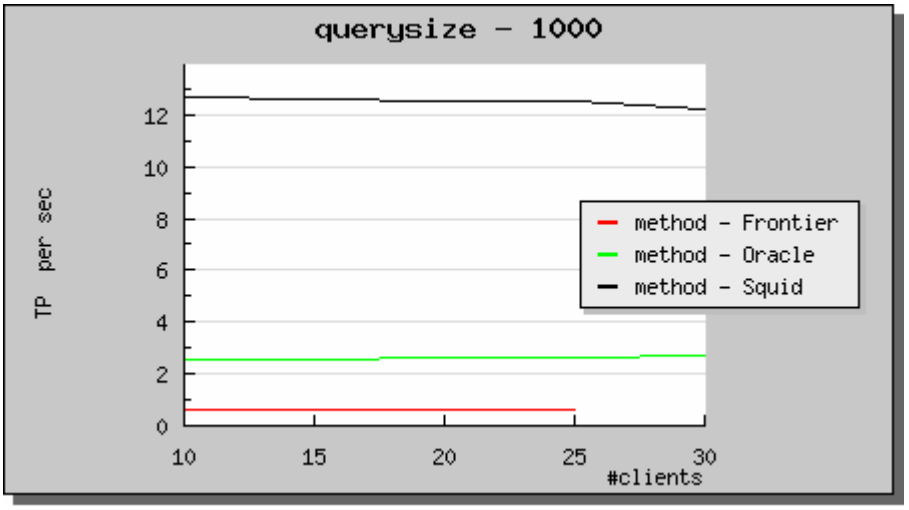


Figure 9 – Access method comparison with 1MB queries and 5 client nodes (throughput / number of clients)

With a bigger number of clients, the behavior for all three access methods was the same as in experiment 1.1. Here we can see that all values for all access methods are stable which means at least one of the system components is saturated:

- Squid access stabilizes at 12MBps with saturated network;
- Oracle access gets to 2,5MBps with saturated database CPU;
- FroNtier access gets to 600kBps with saturated FroNtier server CPU.

The following additional conclusions can be taken: FroNtier puts very low load on the client CPU (5%), Oracle puts average low load on the client CPU (20%), and Squid puts considerable load on the client CPU (40%).

Experiment 1 - Questions

How would a combined access (FroNtier + Squid) compare with Oracle for this simple workload, and for simulation or real workloads?

5.2.5 Experiment 2 - FroNtier Server Access Analysis

Description: FroNtier server base performance analysis from the end user perspective
Parameters:

- Data access method - direct FroNtier access

Workload: CORAL
Factors:

- Number of clients with levels 1, 3, 5, 7 (one client node)
- Query size with levels 10kB, 100kB, 1MB
- Compression factor (zip level) with levels 0, 1, 5, 9

Experiment Design: Full factorial design with 2 repetitions
Experiment duration: 300 seconds
Number of experiments: 4 x 3 x 4 x 2 repetitions = 96

Experiment 2 - Conclusions

Resource Usage Analysis

Regarding the tests without compression (using zip level 0), when queries are small (10kb), the bottleneck is the database CPU with very high utilization. Nevertheless, with bigger query sizes (100kB and 1000kb) there is no apparent bottleneck although high consumption of DB CPU, client CPU, and network is observed.

For tests using compression, it is obvious that the bottleneck is the FroNtier server CPU with utilizations over 90% (result set compression consumes much of the FroNtier server CPU). The only exception is when the compression factor is 1 (less and faster compression) and the query size is 10kb where the FroNtier CPU utilization falls to 70% and the DB CPU utilization grows higher.

The following plot shows the aggregate throughput (in rows per second) and CPU utilization on the FroNtier server per number of clients when executing 100kB direct FroNtier access queries using a compression level of 5.

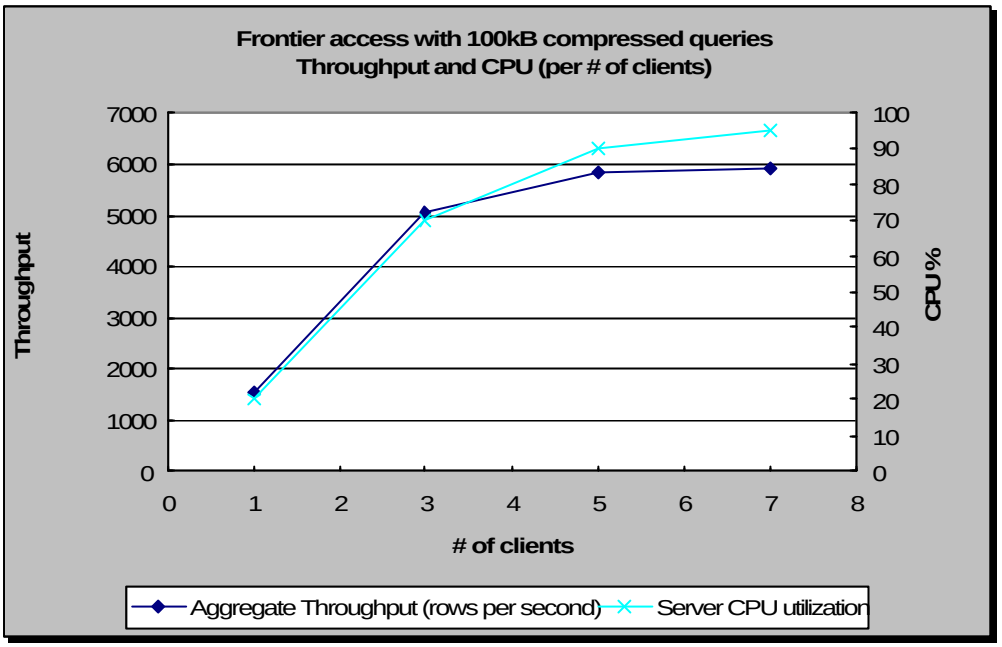


Figure 10 - FroNtier analysis with compressed 100kB queries (throughput and CPU% / number of clients)

In this plot, the CPU utilization on the FroNtier server and the throughput grow with similar rates, and, most importantly, we can see the throughput reaching its maximum at 6000 rows per second (almost 600kBps) when the CPU utilization is reaching the 100%.

The following plot shows the throughput and load average plotted against the CPU utilization for the same test with 100kB compressed queries.

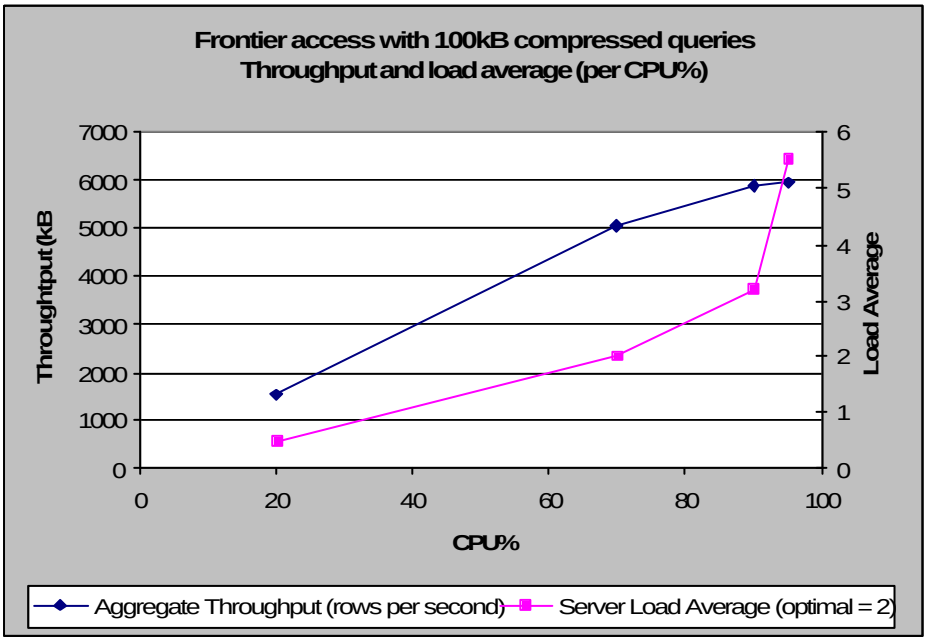


Figure 11 – FroNtier analysis with 100kB compressed queries (throughput and load average / CPU%)

In this plot, we can see the growth rate of the throughput relative to the CPU utilization, i.e., the throughput grows constant with the CPU utilization. The load average grows exponential relative to the CPU utilization and gets up to 6 (the optimal value, for this two processors server, is 2).

Some conclusions can be taken from this experiment in terms of resource utilization:

- Bigger query sizes and higher compression factors imply higher use of FroNtier CPU;
- Higher use of FroNtier CPU implies less use of the DB CPU (as the global throughput is lower);
- The higher the compression factors the faster the bottleneck is reached.

Throughput Analysis

The following plot shows the throughputs for different number of clients and different compression factors (zip levels) for 1MB direct queries to the FroNtier server.

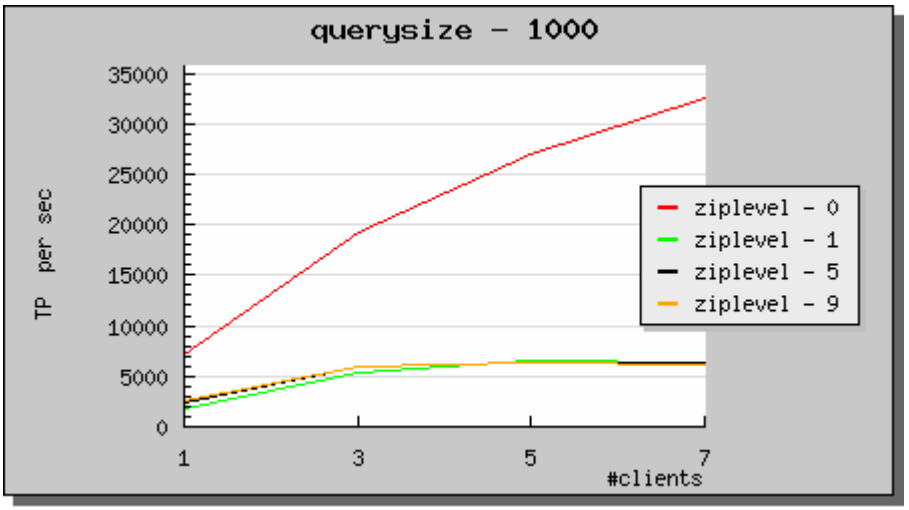


Figure 12 - FroNtier analysis with 1MB queries and different zip levels (throughput / number of clients)

With this specific payload (1MB CORAL direct queries to the FroNtier server), zip level 0 (no compression) performs at least 5 times better than the other zip levels. Additional, all compressing alternatives show similar performances.

The following plot is similar to the previous but the query size is 10kB instead of 1MB.

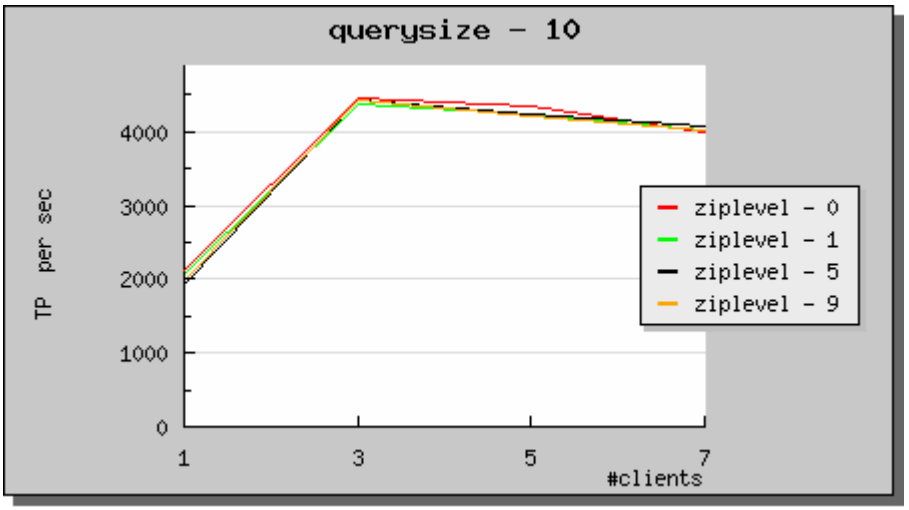


Figure 13 - FroNtier analysis with 10kB queries and different zip levels

(throughput / number of clients)

With 10kB queries, the performance is very different and stands as an exception to bigger query sizes. In this case the performance is the same for all zip levels. This exception is explained by the fact that the non compressible overhead of all packages (where the result set is encapsulated) and the network overhead are of the same order of magnitude of the actual payload data. The throughput degradation for more than 3 clients is again explained by the FroNtier and network overhead that affect the performance.

The following plot shows the throughputs for different query sizes and different compression factors (zip levels) when 7 clients are executed against the FroNtier server.

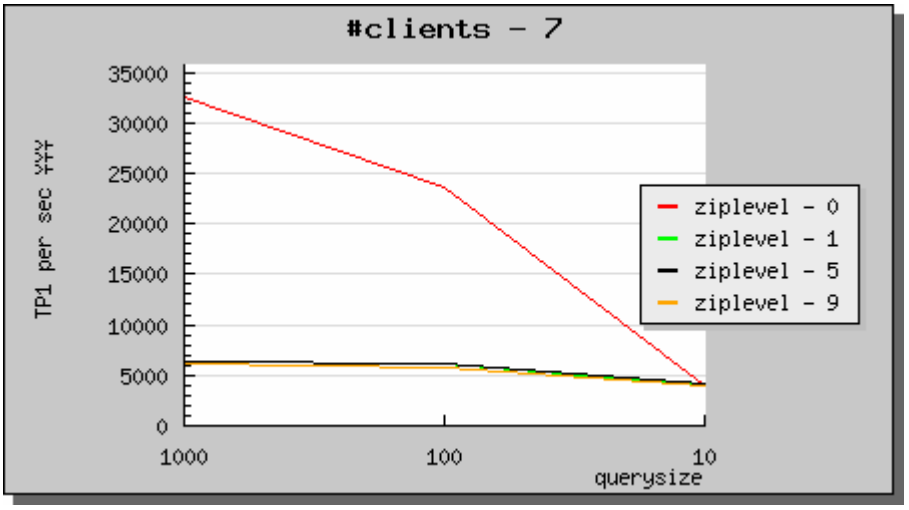


Figure 14 - FroNtier analysis with 7 clients and different zip levels (throughput / query size (kB))

As we can see in this plot, the bigger the query size, the bigger the difference between no compression and all compressing zip levels (granted that the experiment is done below the network bottleneck value).

From these previous plots, we can conclude that payload compression introduces a great load on the FroNtier server and degrades the performance, granted that a good network connection is available. This reveals the importance of understanding if payload compression is worth it. This will be have to be studied using different payloads that will, in some cases, overload the FroNtier CPU and, in other cases, the network bandwidth.

Compression Level Analysis

The following plot shows the throughputs for different number of clients and different compression factors (zip levels) when 1MB queries are executed against the FroNtier server.

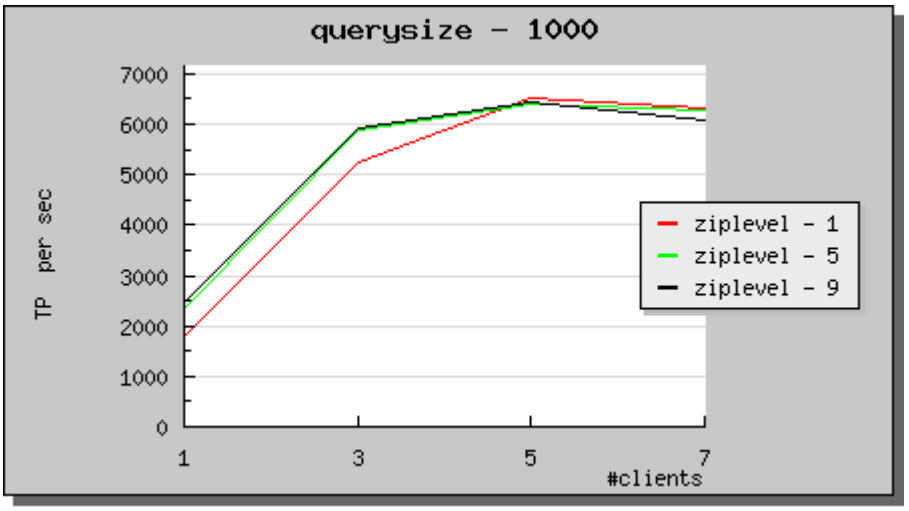


Figure 15 - FroNtier analysis with 1MB compressed queries and different compressing zip levels (throughput / query size (kB))

For all query sizes (1MB query size is depicted in the previous plot), 5 clients (running on one client node) are enough to get to the maximum throughput, with the FroNtier server CPU consumption as the bottleneck.

Zip level 1 is worse for few clients, as there is enough available CPU on the FroNtier server to compress result sets efficiently. Nevertheless, for bigger number of clients (where FroNtier server CPU is much more loaded), zip level 1 performs better than other zip levels as it better uses the short FroNtier server CPU resources. This can be seen in the previous plot where zip level 1 (red line) starts with worse throughput and, as the number of clients grows, gets higher throughput than the other two zip levels.

Experiment 2 – Questions

How does FroNtier perform with more clients using zip level 0?

5.2.6 Experiment 2.1 - FroNtier Server Analysis with No compression

Description: FroNtier server analysis with zip level 0 (no compression) and many clients

Parameters:

- Data access method - direct FroNtier access
- Compression factor (zip level) - 0

Workload: CORAL

Factor:

- Number of clients with levels 5, 10, 15, 20, 25 (5 client nodes)
- Query size with levels 10kB, 100kB, 1MB

Experiment Design: Full factorial design with 2 repetitions

Experiment duration: 300 seconds

Number of experiments: 5 x 3 x 2 repetitions = 30

Experiment 2.1 – Conclusions

Resource Usage Analysis

For all query sizes, this experiment puts low load on the client node as, even running 5 clients, each node shows a CPU utilization of 20% and a load average lower than 0,5.

In the FroNtier server, one can observe a rather high network activity (proportional to the throughput) of up to 6MBps. The CPU utilization is proportional to the number of clients running and to the query size; it grows from 10% to 70% and the load average from 1 to 3 (25 clients with 1MB queries).

In the database server we can observe network traffic of up to 5MBps and a CPU utilization that is inversely proportional to the load on the FroNtier server, i.e., the bigger the query and the number of clients, the higher load on the FroNtier server and less load on the DB. The CPU utilization on the DB server goes from 65%-90% (10kB queries) to 20-30% (1MB queries), the load average goes from 1 up to 5 (10kB queries).

Although all the utilization values are very high, these values indicate that no single resource holds as the bottleneck. In this case, we can consider that the system is saturated as a whole.

Throughput Analysis

The following plot shows the throughputs for different number of clients and different query sizes using zip level 0.

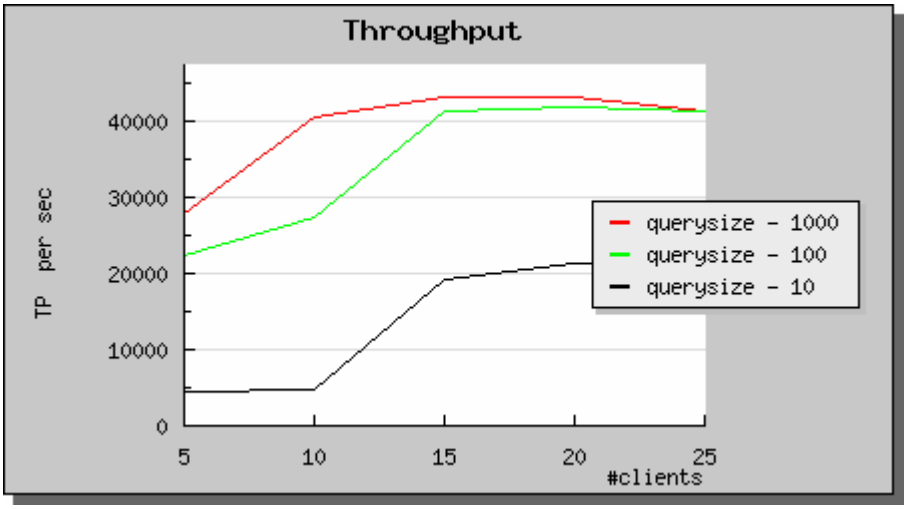


Figure 16 - FroNtier analysis with different sizes of compressed queries (throughput / query size (kB))

In this experiment, we tried to force the limits of the FroNtier server working with no compression. In Figure 12 we could not saturate the system with no compression as the throughput was still going up even with 7 processes running in one client node. From the previous plot we can see that, for all query sizes, 15 clients (3 processes in each of the 5 client nodes) is the point where the maximum throughput is achieved. For example, with queries of 1MB, using zip level 0 (no compression), the clients achieved an aggregated throughput of 40000 rows per second, i.e., almost 4MBps, while other zip levels get only to 600kBps. This represents a factor of more than 6 between compressing and non compressing zip levels.

Compressing the result sets in the FroNtier server aims to improve the throughput by assuming the bottleneck is the network bandwidth. The numbers obtained in this experiment should induce some further investigation of the big impact of compression on a FroNtier setup.

Experiment 2.1 - Questions
What is the relation with zipping performance and compressibility?

5.2.7 Experiment 2.2 - Compression and Compressibility

Description: zip levels analysis with different compressibility levels of the workload
Parameters:
- Query size - 1MB

Workload: CORAL
Factor:
- Number of clients with levels 5, 10, 15, 20 (5 client nodes)
- Compression factor (zip level) – 0, 1, 5, 9
- Query data compressibility with levels 5, 30, 50, 70, 90%

Experiment Design: Full factorial design with 1 repetition
Experiment duration: 300 seconds
Number of experiments: 4 x 4 x 5 x 1 repetition = 80

Experiment 2.2 – Observations: a word about compressibility

Data compressibility is a very important factor in this FroNtier performance analysis. High compressible data bursts the use of compression algorithms in the FroNtier server as the time lost in the FroNtier server CPU compressing the data will represent a considerable gain in the network utilization. Conversely, low compressible data makes compression a waste of time as the time lost in the FroNtier server compressing data will not influence the network bandwidth utilization.

In practical terms, the main question in this analysis is: how compressible will be the real data produced by the accelerator systems at CERN? And the follow up question is: what data should be used to analyze the FroNtier system?

As real data is still not available for testing, only two data sources could be used for this analysis: random data or simulation data. Simulation data is generated by the accelerator simulation programs. At the time of this study, it was not possible to obtain usable simulation data. Nevertheless, some insights about this data were obtained; most importantly: experts stated that simulation data has average compressibility. So, if we take 50% compressibility as the simulation data compressibility, we can use this value as input for the random data generation.

Although several random data generators exist, the obvious choice was the Oracle database random data generator as it is very conveniently available inside every Oracle database.

To achieve low compressible data in an Oracle database table, not only the data must be random (supplied by the Oracle random number generator) but it must fill all the spectrum of representability of the data type in use, i.e., randomness depends on what data type is used to the store the generated values. Random numbers should spread the full possible values of a storage data type, i.e., random numbers size must be the size of the storage field. In order to

fill a 32bit integer field one has to use random numbers of 32bits. If 16bits are used in a 32bit field, the data will be highly compressible, although random numbers are used. Floating point numbers offer a special challenge as the generated numbers must cover the complete range of possible floating point numbers, which is not sequential.

Experiment 2.2 – Conclusions

Resource Usage Analysis

In this experiment, resource usage at DB and client level is always comfortably below the maximum capacity. The bottleneck here is always the FroNtier server CPU: maximum throughput is typically achieved with 100% CPU utilization and load average of 8.

Throughput Analysis

The following plot shows, for the different data compressibility levels and compression level 0 (no compression), the different data flows at different points of the system.

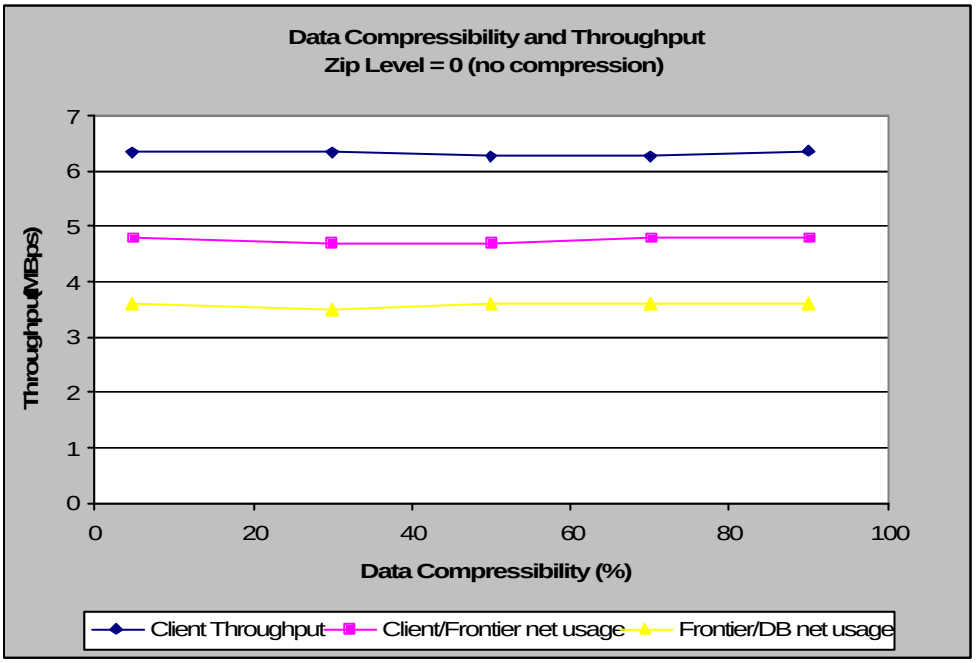


Figure 17 – Data compressibility and Throughput with no compression (throughput / compressibility(%))

In this plot, the blue line shows the throughput seen by the clients (the real data throughput), the pink line shows the network usage between the clients and the FroNtier server (XML documents), and the yellow line the network usage between the FroNtier server and the database server (standard JDBC connection).

As expected, we can see that, with no compression, compressibility and throughput are independent. As the used compression level is 0 (no compression), the network usage between the clients and the FroNtier server and the network usage between the FroNtier server and the database server are both constant, and so, independent from the compressibility.

The network usage between the clients and the FroNtier server (around 4.8MBps) is higher than the network usage between the FroNtier server and the database server (around 3.6MBps) due to character encoding techniques. The connection between the FroNtier server

and the database is a standard JDBC. The connection between the clients and the FroNtier server is based on XML documents generated by the FroNtier server that encodes the result sets using the base64 encoding. Base64 is an ASCII based representation that uses 4 bytes to encode 3 bytes (6 bits from each of the 4 bytes). This encoding is safer to transmit binary data inside the XML files as it only uses readable ASCII character codes to represent binary data. Conversely, it uses more space to represent the same binary data. Base64 is theoretically 25% less efficient (it uses 4 bytes to encode 3 bytes) which is exactly what we can see on the previous plot: 4,8MBps of JDBC binary data and 3,6MBps of base64 data on XML files (75% of 4,6MBps) that flow between the client and the FroNtier server.

The following plot shows the same information as the previous plot for the test case with compression level 1, which is the lowest compression level.

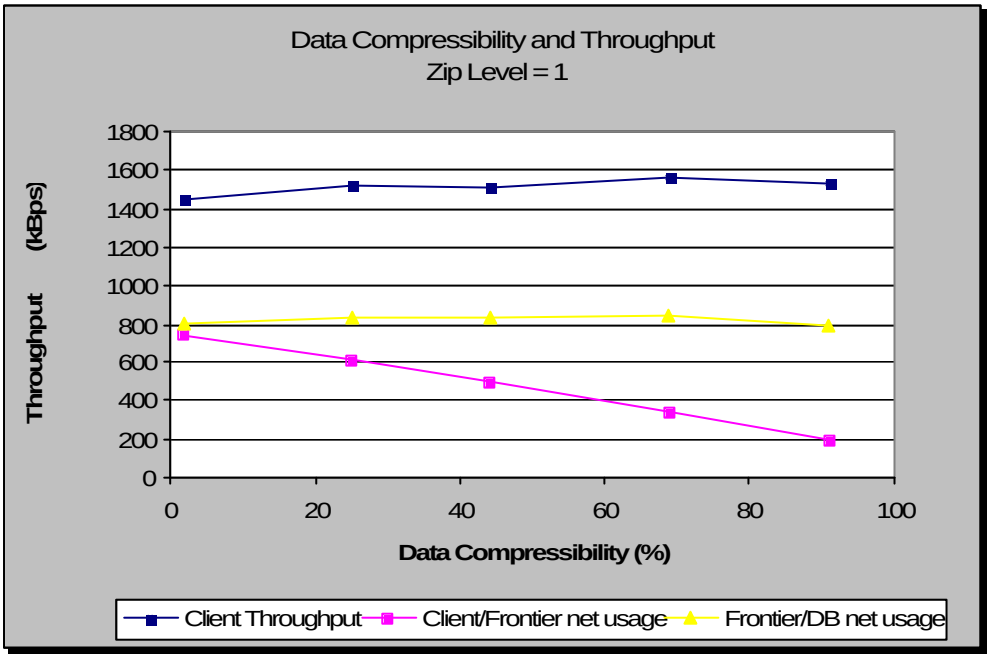


Figure 18 – Data compressibility and Throughput with compression level 1 (throughput / compressibility (%))

The meaning of the lines on this plot is the same as on the previous. In this plot, the values are much lower (expressed in kBps). Maximum client throughput is around 1500kBps instead of 6,3MBps (with no compression). This is explained by the fact that this test case forces a high FroNtier CPU load (which is the typical bottleneck when using compression).

The most interesting detail in this plot is the network usage between the clients and the FroNtier server (pink line in the plot) that goes down with compressibility. The compression is only seen here, the JDBC connection between the FroNtier server and the database is not compressed. The more compressible the data is, the lower is the network usage between the clients and the FroNtier server. If we draw a line (following the pink line) to the 100% compressibility, we see that 50kB of network traffic is the point at which 100% compressibility would be reached. So, for 1MB queries, there is 50-100kB of base network traffic being used (base overhead), which is 5-10% of the query size.

Nevertheless, the most important conclusion here is that the client perceived throughput is rather stable and independent from the compressibility of the data, provided we stay close to the FroNtier CPU maximum load.

The next plot presents the throughput achieved in this experiment with different number of clients and different compression levels using 30% compressible data.

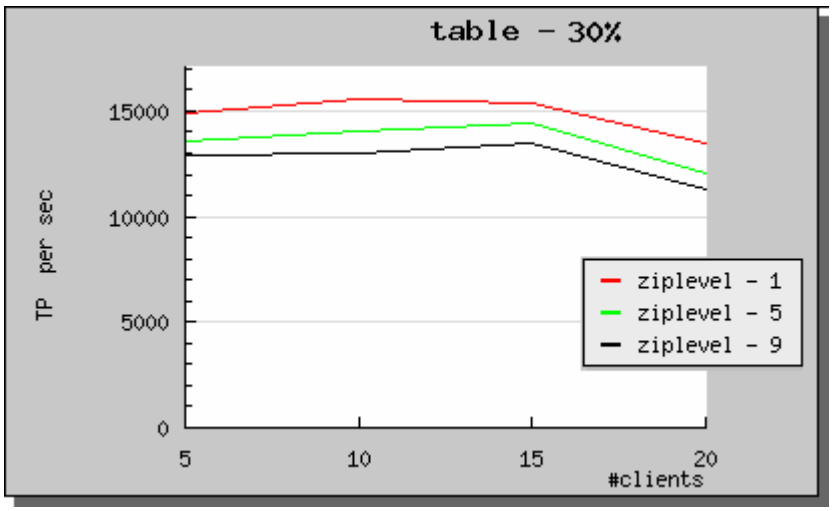


Figure 19 – FroNtier analysis with 1MB queries against a table with 30% compressible data (throughput / number of clients)

This plot is equivalent to the plot in Figure 15 and works as a confirmation of the conclusions taken there. In this case, as more clients are used (5 client nodes), all the values are very close to the maximum throughput as the CPU usage in the FroNtier server is always the maximum.

As we are testing 30% compressible data, compression level 1 (red line) has the best performance and gets to the maximum throughput with less clients (10). We can conclude that using higher compression levels and make an extra effort in trying to compress not very compressible data (30%) is useless in this case.

The same experiment executed with more compressible data (not shown in this plot) shows compression level 5 performing as good as compression level 1 while compression level 9 is always the worse compression level.

The compression level is a system parameter that will be defined and used always with the same value. As compression level 9 always performs worse than the others (even with highly compressible data) and compression level 5 only performs as good as compression level 1 in highly compressible data, we can conclude from this experiment that compression level 1 is most probably the best choice as most of the real data used in the project will not be highly compressible.

5.2.8 Experiment 2.3 – Big Queries Analysis

Description: FroNtier server analysis using query sizes of up to 30MB
Parameters:

- Compression factor (zip level) – 5
- Query data compressibility - 50%

Workload: CORAL
Factor:

- Number of clients with levels 5, 10, 15 (5 client nodes)
- Query size with levels 1, 5, 10, 20, 30 MB

- Data access method - direct FroNtier access and Squid cache access

Experiment Design: Full factorial design with 2 repetitions
Experiment duration: 900 seconds
Number of experiments: 3 x 5 x 2 x 2 repetition = 60

Experiment 2.3 – Observations: real world query sizes

Experts state that real scenario normal queries will be around 2-3MB, while small queries will be 100kB and bigger queries will reach 20-30MB. Until this point, tests were executed with query sizes of 10kB, 100kB, and 1MB. We have seen that 10kB have a rather particular behavior as their size is of the same order of magnitude as most of the overheads. Based on the experts' statement, we should ignore the issue with 10kB queries and further analyze the behavior of the FroNtier server with queries of 20-30MB.

Experiment 2.3 – Conclusions

Executing this test against a Squid cache proved that big queries are not a problem for Squid access. Using the Squid cache, the network bottleneck is reached in all cases, limiting the maximum throughput. Moreover, as the data used in this experiment is 50% compressible and the network link available is a 100Mbps connection, we could predict a maximum throughput of around 25MBps, equivalent to 12,5MBps (100Mbps) times 2 (50% compressibility): 20MBps were achieved.

Regarding the execution of the test against a FroNtier server using compression level 5, the following plot shows the achieved throughput for each query size.

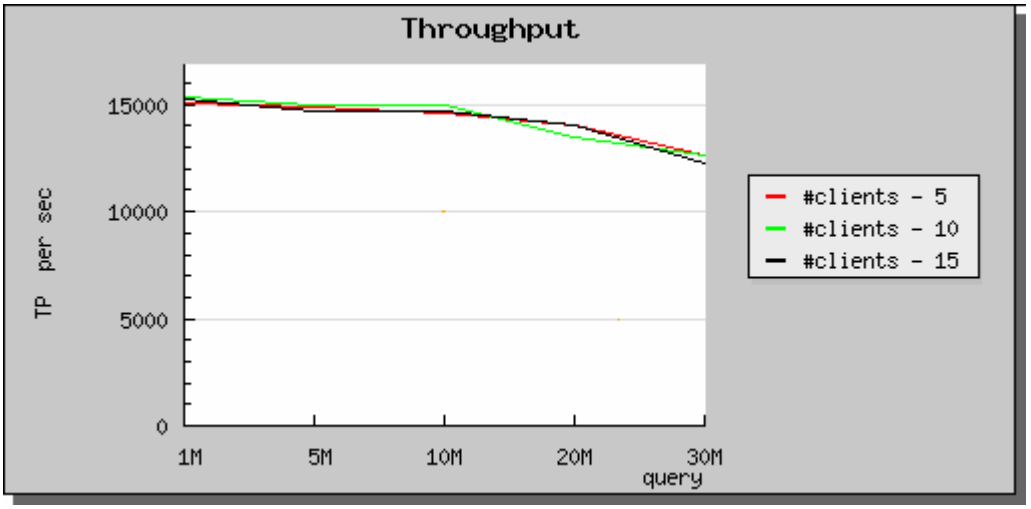


Figure 20 – FroNtier analysis with big compressed queries (throughput / query size (MB))

In this plot, we can see that the throughput behavior is the same for all number of clients. The resource utilization is also the same as in other similar experiments: the CPU utilization on the FroNtier server is very high (near 100% with loads of up to 10), and the CPU utilization on the clients is rather low.

Additionally, for bigger number of clients (20 or 25), the aggregate throughput is lower than the throughput achieved with 5, 10, and 15 clients. Two base reasons exist for this: for bigger number of clients, each dient request is longer, as more clients are saturating the system, and so, the test duration should be bigger than 900 seconds (15 minutes). On the

other hand, testing with such a big number of clients (for just one FroNtier server) makes server errors to be much more probable, which limits the throughput.

As a general conclusion, we can state that the FroNtier server can handle queries of 20MB-30MB with no issues. Nevertheless, the behavior of the FroNtier server under extremely heavy load (bigger queries, bigger number of clients, and bigger test durations) should be tested.

Experiment 2.3 – Questions

How does FroNtier server works under extremely heavy load? What is the error rates progression? What is the capacity of a FroNtier server in terms of number of clients?

5.2.9 Experiment 2.4 – Error Rate Analysis

Description: FroNtier server capacity and error rates analysis

Parameters:

- Compression factor (zip level) – 0
- Data access method - direct FroNtier access
- FroNtier server distribution - 3.1

Workload: CORAL

Factor:

- Number of clients with levels 1, 5, 10, 20, 30, 40, 50, 80, 100, 150 (10 client nodes)
- Query size with levels 1.3, 2.7, 5.5, 11.2, 22.6 MB

Experiment Design: Full factorial design with 1 repetition

Experiment duration: 1800 seconds (30 minutes)

Number of experiments: 10 x 5 x 1 repetitions = 50

Experiment 2.4 – Conclusions

Test duration should be carefully controlled as, if a short duration is used, the obtained values will not be stable and will show a great variability. In this experiment, with many clients causing many errors and long response times, the test duration used was 30 minutes (for each combination of levels).

The following plot shows the error rates (% of failed queries) for different query sizes and number of clients. In it, we can see that the bigger the query size, the sooner the error rate starts to grow, i.e., less number of clients are needed to saturate the server. If we consider 10% as the threshold for the maximum acceptable error rate, we can define the maximum capacity for each query size for a single FroNtier server. For example, the server can handle (with error rates below the defined acceptance threshold of 10%), 80 clients issuing queries of 1,3MB, 30 clients issuing queries of 5,5MB, etc. Moreover, real clients will have a much slower query frequency, thus, the real capacity of the server is expected to be quite higher than these values.

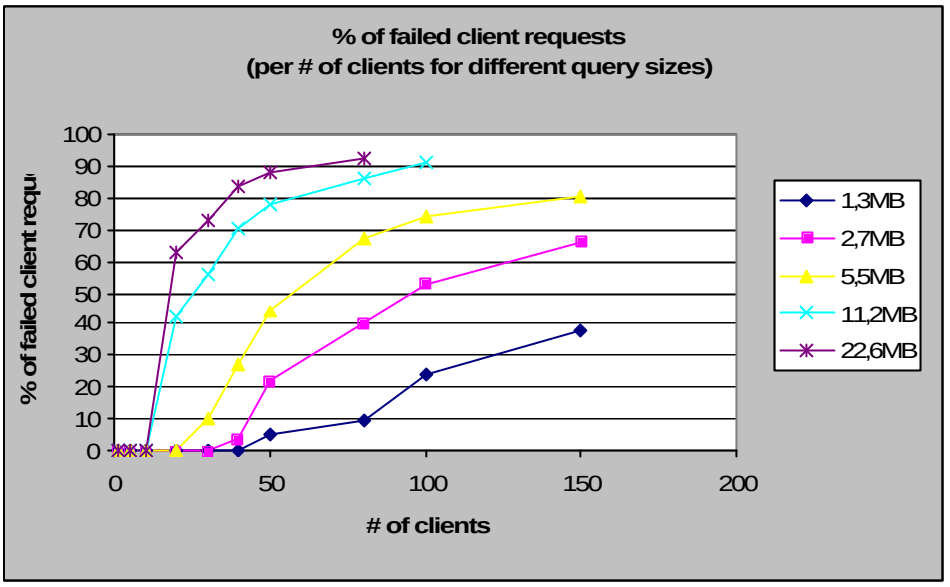


Figure 21 - % of failed queries with different query sizes and different number of clients (% of failed queries / number of clients)

As a detailed example, in the next plot we can see the error rate and the aggregated throughput for queries with 2,73MB and different number of clients.

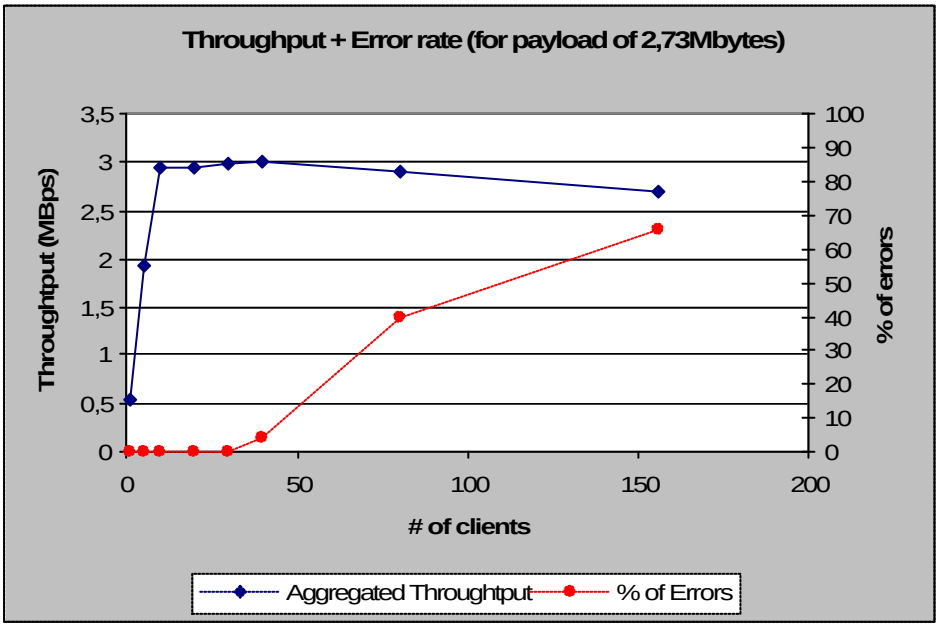


Figure 22 - Throughput and % of failed queries for 2,73MB queries with different number of clients (MBps and % of failed queries / number of clients)

In this plot, we can see the maximum aggregated throughput being achieved with 5 to 10 clients and then the error rate starting to climb with 30/40 clients. Again, if we define the acceptance error rate threshold as 10%, we can see that the maximum capacity for this setup would be around 40/50 clients.

When many clients are running against a single server and the error rate is high (as we can see on this plot with more than 50 clients), aggregate throughput goes slightly down as error requests are also processed (before the error occurs) and do take server CPU time.

5.2.10 Experiment 3 - Squid Cache Access Analysis

Description: Squid cache base performance analysis from the end user perspective
Parameters:
- Data access method – Squid cache access

Workload: CORAL
Factors:
- Number of clients with levels 4, 8, 12, 16 (4 client nodes)
- Query size with levels 10kB, 100kB, 1MB
- Compression factor (zip level) with levels 0, 1, 5, 9

Experiment Design: Full factorial design with 2 repetitions
Experiment duration: 300 seconds
Number of experiments: 4 x 3 x 4 x 2 repetitions = 96

Experiment 3 – Conclusions

Resource Usage Analysis

In this experiment, the aim is to analyze the network usage between the servers and the CPU utilization on the client nodes. As there is no compression work on the FroNtier server (the documents are already compressed in the cache) and almost no access to the database is done (except FroNtier metadata queries), the resource utilization is very low except on the network and on the client CPU (where the documents are uncompressed).

When no compression is used, the bottleneck of the system is always the network usage. In all cases, the network usage on the FroNtier/Squid server reaches 11,5MBps (96Mbps) which is very close to the theoretical 100Mbps limit of the network. For example, with 4 clients, each client gets to 3MBps of network usage. The CPU usage on the FroNtier/Squid server is 10% that represent the Squid server servicing cached queries and the FroNtier server servicing the metadata queries. Although the client CPU is not a bottleneck on this case, it gets to 70% with 10kB queries and to 30-50% with bigger queries.

When compression is used, the FroNtier/Squid CPU load is very low. As in experiment 1.1, with small queries of 10kB, although the network usage is high (8MBps), the bottleneck is the client CPU that cannot handle so many small queries to uncompress and shows a CPU utilization of over 90%. On the other cases, the bottleneck is the network usage between the FroNtier/Squid server and the clients (11,5MBps).

In this experiment, there is almost no activity in the DB server as most of the queries are cached. Nonetheless, FroNtier metadata queries issued by the client are not cached on the Squid server. For each user query, FroNtier/Squid has to execute a certain number of metadata queries to the database. These queries are queries to the Oracle data dictionary. As these are the only queries that are not cached in the Squid server, the traffic generated by these metadata queries can be measured on the network usage between the FroNtier server and the database. For 10kB queries, the network traffic between the database and the FroNtier/Squid server is 800kBps; for 100kB queries, 370kBps; and for 1MB queries, 280kBps. If the queries are bigger, this network usage between the FroNtier server and the

database is lower. To achieve similar throughput rates (limited by the network between the FroNtier server and the clients) with smaller queries, a much bigger number of queries has to be executed. This is why the database network usage goes down when the query size grows.

It is also important to understand the different types of overheads that exist in the system as compressed payloads always include non compressible overheads. For example, in a query of 10kB, overhead size can reach 50% of the transmitted data, and so, drastically influence the performance. The types of overheads are: database overhead (each element on the database has a storage overhead, for example, a row as a 4 bytes overhead), the network overhead (network protocols overhead), FroNtier client overhead (xml documents communication management like error control), CORAL FroNtier plug-in overhead, etc.

Throughput Analysis

The following plot shows the throughput achieved with different number of clients and different compression levels for 1MB queries.

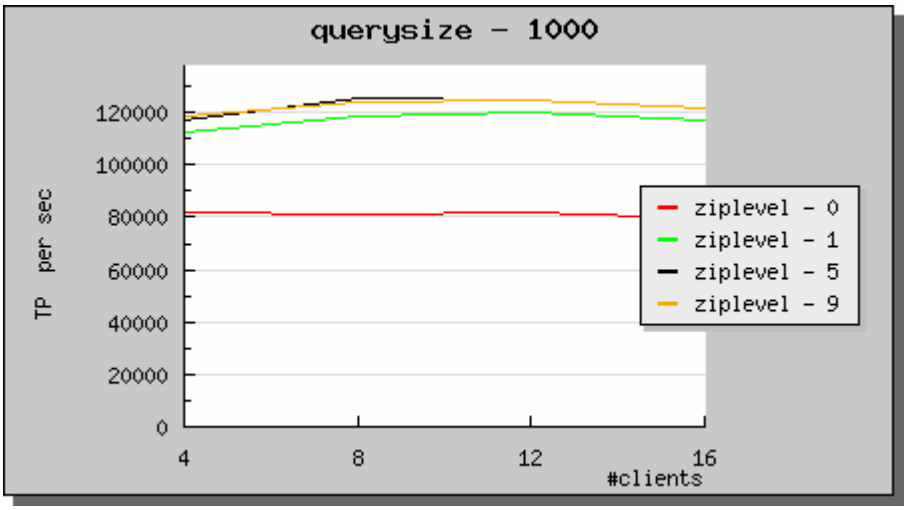


Figure 23 - Squid analysis with 1MB queries and different compression levels (throughput / number of clients)

The first detail we should point out is the fact that cached Squid access is faster using compression (red line appears below all the other lines). This is a basic fact about FroNtier: no compression is faster when accessing FroNtier directly (Figure 12) and slower when accessing Squid cached results (Figure 23). On Figure 12, when accessing the FroNtier server directly, the bottleneck is the FroNtier server CPU, and so, no compression is faster as it puts a lower load on the FroNtier server. On the other hand, in this plot, when accessing Squid cached results, the bottleneck is the network usage and so, compressing the payload is advantageous to explore the heavy loaded network link. If we assume that the real workloads will be based on very repetitive access patterns, we can conclude that using compression on the FroNtier server is advantageous.

Although, in this experiment, we used almost non compressible data, all workloads include compressible metadata which explains the small difference between compression and no compression. If the data was more compressible, compression tests would perform even better than no compression.

In this experiment, we could observe the exact same behavior as depicted in Figure 7 where tests with 10kB queries would perform much worse than bigger queries due to the size of the communication and metadata overhead.

5.2.11 Experiment 4 - COOL Workload Analysis

Description: FroNtier, Squid, Oracle performance analysis from the end user perspective
Parameters:

- Query size – 10000 database table rows of 159bytes each – 1,52MB

Workload: COOL
Factors:

- Number of clients with levels 4, 8, 12, 16 (4 client node)
- Data access method with levels direct FroNtier access (FroNtier), Squid cache access (Squid), and direct database access (Oracle)
- Compression factor (zip level) with levels 0, 1, 5, 9 (for Squid and FroNtier only)

Experiment Design: Fractional factorial design with 2 repetitions – level Oracle of access method factor is not mixed with zip levels (direct Oracle access cannot be compressed)
Experiment duration: 300 seconds
Number of experiments: 4 x (1 [Oracle] + 2 [FroNtier and Squid] x 4) x 2 repetitions = 72

Experiment 4 – Observations

This experiment uses the COOL workload as described in section 4.8. This workload is used with the primary objective of validating the experiments done with the base workload (CORAL) as it represents a step further on the test of the FroNtier software stack. The aim is to find special behaviors that would not be seen using the base workload (CORAL).

This COOL workload is based on queries captured from a typical utilization of the COOL API. It consists of a list of COOL queries that are executed against a COOL database structure filled with random data.

Experiment 4 – Conclusions

The following plot shows the throughput achieved with the different access methods and compression levels for different number of clients.

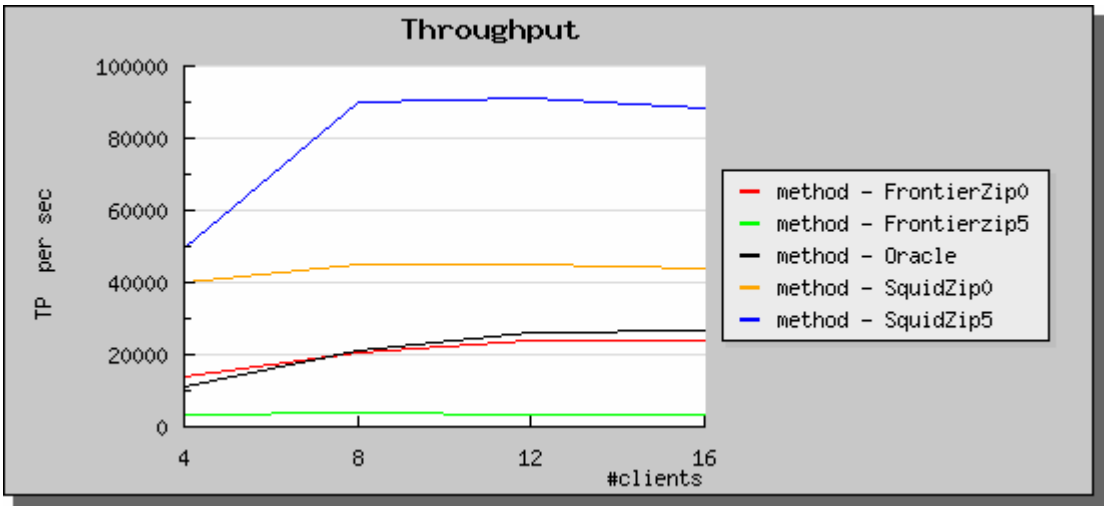


Figure 24 – COOL workload analysis with different access methods and compression levels (throughput / number of clients)

This plot is very similar to Figure 7 where the performance of the various access methods is depicted. On this plot, the compression levels are added to the access with FroNtier and Squid.

Oracle access (black line) and FroNtier access with no compression (red line) have the same behavior as the FroNtier server has only to forward the database results to the clients. Nevertheless, a small degradation of FroNtier access is noticed with 12 and 16 clients where direct Oracle access gets naturally faster.

Although Squid cached access (blue and orange lines) performs much better than Oracle and FroNtier (red and black lines), the difference is smaller than when using the CORAL workload (Figure 7). In this case, the COOL workload contains more overheads and metadata queries. Moreover, Squid access with no compression (orange line) losses on overhead compression, coral metadata queries compression, data compression, and, on this specific case, on COOL metadata queries compression.

If we go from the non compressing lines (red and orange lines) to the compressing lines (blue and green lines) we see: FroNtier gets much slower (red to green line) and Squid gets much faster (orange to blue line). This happens because, on the FroNtier side, the FroNtier server gets stuck with the compression tasks and, on the Squid side, the network is much better used with compressed results. This factor is highly noticeable because the data used in this experiment is highly compressible. The combined access question arises again: will the combined FroNtier/Squid access (blue/green lines or orange/red lines) be faster than Oracle access (black line)?

A major conclusion is that the throughputs achieved with this COOL workload are slightly lower than the ones achieved with the CORAL workload. This can be simply explained by the execution of more metadata queries by COOL. The more we move up testing the software stack, the bigger this effect will be.

Apart from these conclusions, no major behavioral differences were noticed between the COOL and the CORAL workload tests. All the factors in this experiment, as data compressibility, number of clients, access method, etc., could be used for further analysis but from this simple experiment we can conclude that no major differences will appear and so, no further tests with the COOL workload will be done.

5.2.12 Experiment 5 – Tier-1 Access Analysis

Description: Analysis of the FroNtier server accessed from a Tier-1 site
Parameters:

- Query size – 1MB
- Test nodes location – Tier-1 site at FNAL

Workload: CORAL
Factors:

- Number of clients with levels 2, 4, 8, 16 (2 client nodes)
- Access method with levels FroNtier access (FroNtier) and Squid cache access (Squid)
- Compression factor (zip level) with levels 0 and 5

Experiment Design: Fractional factorial design with 1 repetition
Experiment duration: 300 seconds
Number of experiments: 4 x 2 x 2 x 1 repetition = 16

Experiment 5 – Observations and Network Tests

This experiment is similar to experiment 1 where CORAL workload is used and different access methods are compared. The particularity here is that the client nodes location is the Tier-1 site FNAL. As this tier-1 site is in the United States, the network link is a transatlantic link and, although it is a good connection, it is naturally slower than the local connections established between nodes at CERN tier-0. The main focus in this experiment is to analyze the impact of this client relocation.

The first test activities performed at FNAL were directly related to network performance. The quality of the network link to CERN was evaluated with standard network performance tools. These first network tests confirmed that connections between nodes at CERN are theoretical and practical 100Mbps connections, and connections between CERN and FNAL nodes are connections with much higher latency, lower performance, and higher performance variability. As a base line, a single node at FNAL connecting to a server at CERN can get a maximum throughput of 2 Mbps (around 265kBps). Tests with more nodes at FNAL showed that this throughput can grow over 10Mbps with parallelization as we will see in the experiment comparing FroNtier and Squid access modes.

Experiment 5 – Conclusions

In this experiment, the CORAL workload is exercised using tier-1 nodes at FNAL. The next plot shows the aggregated throughput achieved with different number of clients and different access methods. In this case, the access methods are combined with compression levels, namely, 0 (no compression) and 5 (medium compression).

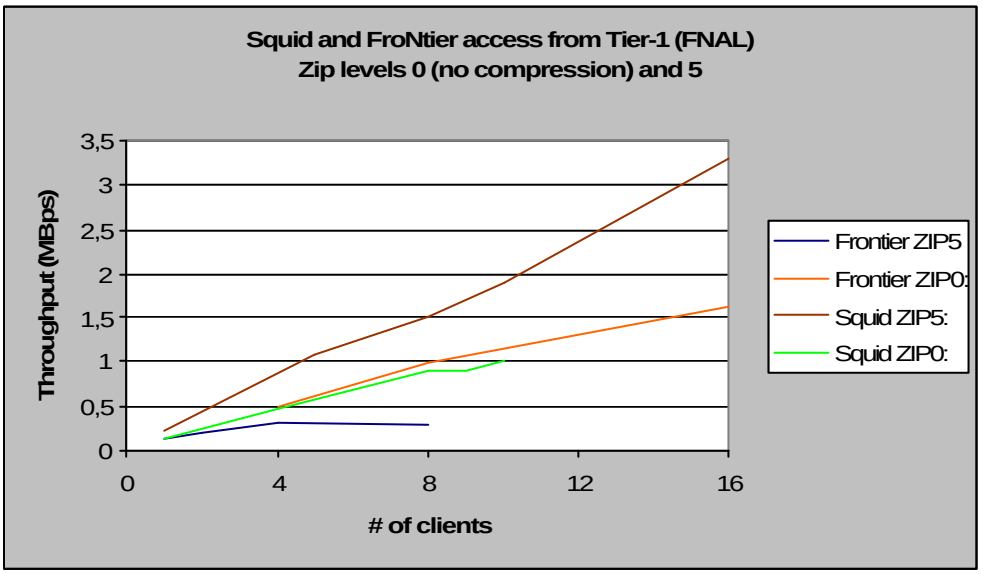


Figure 25 - Tier-1 execution analysis with different access methods and compression levels (throughput / number of clients)

In this plot, we can see that Squid access with compression level 0 (Squid ZIP0) performs almost the same as direct FroNtier access with compression level 0 (FroNtier ZIP0). This clearly tells us that the FroNtier CPU is not the bottleneck of the system. From these two lines we can probably conclude that these values are following the network bottleneck of the system that reaches a maximum of 1,6MBps (around 20Mbps).

On the other hand, we can see the FroNtier access using compression (FroNtier ZIP5) showing the exact same values obtained at CERN. Even from this tier-1 site, a few clients are enough to saturate the FroNtier server CPU that shows very high CPU utilizations. Conversely, the Squid access using compression (Squid ZIP5) shows the benefits of using compression by performing better than the probable network bottleneck of 20Mbps. Squid access with compression can get to 26,4Mbps (seen by the clients) in a 20Mbps network connection. These 26,4Mbps depend on the compressibility of the data.

Although this was a very limited test on the tier-1 site, we could validate the execution of the software on the tier-1 site. Moreover, we could verify that a few clients can saturate the FroNtier server with direct access from tier-1, and that compressing cached documents can be decisive to make the most of a short network connection.

5.2.13 Experiment 6 - ATHENA Workload Analysis

Description: Compare Oracle with FroNtier/Squid by executing the Athena workload
Parameters:
- Query size - 1MB
- Compression factor (zip level) – 5

Workload: ATHENA
Factors:
- Number of clients with levels 17, 34, 51 (17 client node)
- Data access method with levels Squid cache access (Squid) and direct database access (Oracle)

Experiment Design: Full factorial design with 2 repetitions
Experiment duration: 1800 seconds
Number of experiments: 3 x 2 x 2 repetitions = 12

Experiment 6 – Observations

This experiment was very useful as its setup enabled a very effective system level debugging. From this setup, some bugs and deployment issues were detected and solved. One of the most important issues was detected in the interaction between the CORAL FroNtier plug-in and the FroNtier client where a lot of validation queries were seen and considered useless. These queries were simple empty queries to the database to test the presence of the database. It was quite a big performance issue as there were between 4 and 6 of these queries for each of the user queries.

Experiment 6 – Conclusions

This experiment has one major factor with two levels: the access method with levels Oracle and Squid. We will compare the execution of the Athena job against the Oracle database and its execution against the Squid server.

Resource Utilization

In terms of resource usage, the execution against a Squid server loaded the network connection between the clients and the Squid server to a medium level below the bottleneck (maximum of 4MBps) while the CPU on the Squid server had 10% utilization. The database server had high CPU utilization (90% - load of 2 to 6) but a very network usage (maximum of 80kBps).

On the test case run against the Oracle database, the database was very loaded on both the network (maximum of 4,5MBps) and on the CPU, with over 90% utilization and load averages of up to 30.

The CPU on the client is, in both cases, under the bottleneck level and is, due to the FroNtier payload decompression, less loaded using Oracle direct access.

The most stressed resource in this experiment is the database CPU. While directly accessing Oracle puts a very high load on the database with very high network throughput, accessing the database through the Squid server puts the database in a high load (but not as high as in the other case) with a low network throughput. This low network utilization (compared to the high load on the CPU) is due to the very high number of very small queries (validation and meta-data queries issued by the Squid cache to the database).

Throughput Analysis

The following plot shows the aggregated throughput (in finished jobs per second) obtained for the direct execution of the job against the Oracle database and the execution against the Squid server.

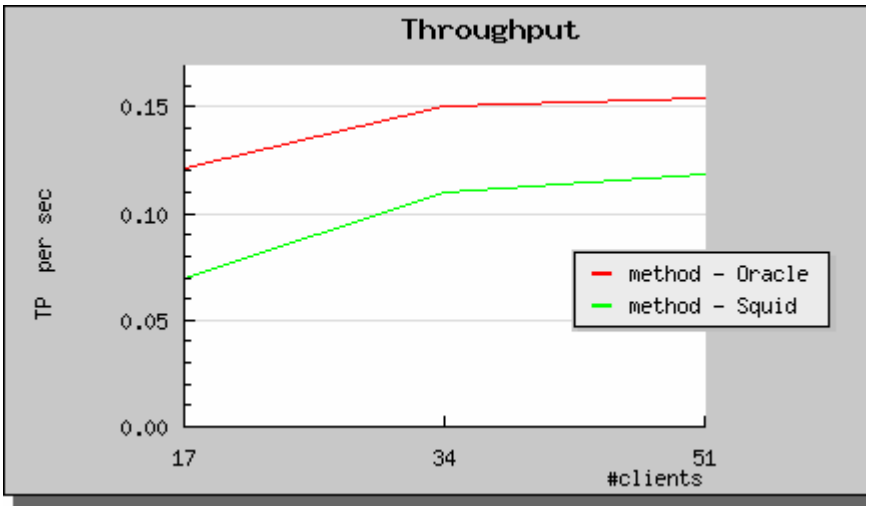


Figure 26 – Athena workload analysis using Oracle and Squid methods (throughput / number of clients)

The main observation to this plot is that Oracle direct access (red line) performs better than Squid cache access (green line). This was not the expected result as Squid cache should show better performance by using cached queries. Nonetheless, this result is easily explained by the existence of extra queries executed by FroNtier CORAL Plug-in (validation queries) and by the many not cached metadata queries.

Oracle access and Squid access represent two methods of data access that, as seen before, have different potential bottlenecks, i.e., Oracle access method will bottleneck on the database server CPU or disk utilization, while Squid access will bottleneck on the network utilization. As this experiment is executed on Tier-0 (the clients are very close to the servers), the network connection are very fast and so, Oracle access can perform better than Squid access.

Oracle direct access (red line) seems to scale more poorly than Squid access. This is an expected result as Oracle direct access as to deal with all client queries while Squid access can reuse the cache for its multiple clients.

It is important to notice here that this experiment is exercising not only the Squid cache in the specific case but also the Oracle database cache. This experiment is exercising only a very reduced part of the spectrum of possible queries and so, the database cache can handle all the queries on this experiment. This is another important reason why the direct Oracle database access is performing better. Nevertheless, in a real scenario, the data content will be much bigger and the data sets will not fit in the database cache, and so, the performance will be quite worse. In the Squid access case, the caches will be spread around different grid sites and the Squid cache data in the disks (not exercised on this experiment) will be used. Moreover, clients on the same site will more likely access the same data sets making the cache hits on the Squid server more probable on the local Squid cache.

This experiment is very important for this study as it is very close to the real utilization. The major difference to the real scenario is that much more clients will be executed. The caching system will be in advantage as the caches will be spread around the grid and, in the database direct access, all the clients will have to access one single central database.

5.2.14 Additional Experiments

This section briefly describes additional experiments with less impact on the conclusions.

CORAL Code Analysis

CORAL Plug-ins control the connections from the clients to the FroNtier server or to the Oracle database (if the Oracle Plug-in is used). In a previous CORAL version, the plug-ins were establishing a connection to the database every time a query was issued and no connection reutilization was done. A version later, a connection pool was introduced and database connections were reused between queries.

In this context, the FroNtier test framework was used to compare the performance impact of this code change; not only on the FroNtier plug-in but also on the Oracle one. The FroNtier test client was used with the CORAL version working as the experiment factor.

The results of this experiment showed that this change had a big impact on the Oracle plug-in performance while on the FroNtier plug-in no performance difference was noticed.

The Oracle plug-in working with the connection pool performs rather faster than the previous version as direct Oracle connections to the database are connection based, so, connection establishment is a considerable overhead. Reusing connection is therefore a performance advantage and performed, in this experiment, 10 to 20% faster.

On the FroNtier plug-in, no differences were noticed. Although CORAL plug-ins abstract layer running on the client is connection oriented, FroNtier is a connectionless system. As the connection to the database is done on the FroNtier server and not on the client, the FroNtier plug-in could not take advantage of this improvement.

Query Frequency Analysis

Queries may either be done at a given rate, to simulate a given flow of information, or continuously, to test the maximum capacity of a system. An average load test (non continuous query rates) was executed to measure the impact difference to the high load tests used throughout this study. In this experiment, each client waits a defined period between requests, instead of being permanently sending query requests to the server.

The results obtained showed that no behavioral difference exists. Naturally, the server capacity in terms of number of clients is higher, as the clients are sending queries to the

server at a slower rate. Nevertheless, if a sufficiently long test duration is used, the aggregate throughput is the same as the throughput obtained in the high query frequency tests.

Tomcat Server Configuration Analysis

A final test compared the performance of the FroNtier server with different tomcat configurations. One of the tested parameters was the validation query parameter in the server.xml configuration file. This parameter defines what validation query the tomcat server should use to validate the presence of the database. In this case the query in use was "select 1 from dual".

In this experiment, the normal CORAL workload was executed against the server running with and without the validation query on the configuration file. The results obtained showed that no performance difference exists, most probably because this query is only executed in the start up of the server.

5.3 Summary

This chapter described in section 5.1 the test setup in terms of hardware and software configuration of the different components of the system: FroNtier servers, database servers, and client servers. In a second part, section 5.2 documented the experiments design, execution, and individual experiment result analysis. Major conclusions are summarized in section 6.1.

Chapter 6 - Conclusions

The LHC, the new particle accelerator at CERN, will start in late 2007. The LHC Computing Grid (LCG) will be used to process all the Petabytes of information that represent the particle collisions happening in real time down in the LHC pits. A considerable part of this information (around 10%) will be stored in a structured form inside databases. The LCG 3D project aims to deploy an efficient way to distribute these databases around all the LCG sites so that scientists around the world can access the LCG databases. This thesis studied the performance of FroNtier, one of the database distribution techniques used in LCG.

This chapter summarizes the major conclusions taken in this work about FroNtier, and compares these results with the results obtained in the deployment and performance tests of Oracle Streams (another database distribution technique). Finally, this chapter presents this thesis' major contributions to the research area and some directions for future work.

6.1 Tests Conclusions

FroNtier

In the previous chapter, the FroNtier performance tests were documented and its conclusions were presented along with the results. The main general conclusions taken from this study are:

- FroNtier server direct access is slower than Oracle direct access and Squid cache direct access is faster than Oracle direct access. When using compression on the FroNtier servers, FroNtier access gets much slower than Oracle access while Squid access gets much faster than Oracle access;
- Payload compression on the FroNtier server is advantageous as it will boost the systems' global performance by efficiently using the short available network bandwidth between the grid sites;
- Compression level 1 is the best choice for the compression level as it takes much less CPU resources on the FroNtier server while compressing the data in a rather efficient way compared to the other compression levels;
- FroNtier can handle queries of 20Mb-30Mbytes with no issues; the FroNtier server used in this study can handle, with error rates below 10%, 80 simultaneous clients issuing queries of 1,3MB, 50 clients issuing queries of 2,73MB, and 30 clients issuing queries of 5,5MB. Real clients will have a much slower query frequency, thus, the real capacity of the server is expected to be quite higher than these values;
- Using the CORAL/FroNtier framework to access structured conditions data with COOL does not generate any performance problem;
- Tests performed from a tier-1 site (transatlantic connection) confirmed that a few clients can saturate the FroNtier server with direct access from tier-1, and that compressing cached documents can be decisive to make the most of a short network connection;
- Tests performed at Tier-0 with the Athena workload, a workload very similar to the real scenario, showed that Oracle performs slightly better than the FroNtier/Squid setup. Some issues were detected in the FroNtier/Squid setup using the Athena

workload, as useless validation queries. The resolution of these issues will improve the performance of the system. Moreover, as the Athena workload is executed on other Tiers (farther away from the central database), FroNtier/Squid setup will get comparatively faster than direct Oracle access. Further tests with real scenarios will most probably confirm this statement;

- A FroNtier/Squid setup covering an Oracle database enables that database to handle a much larger number of clients as many clients will not need to directly access the database;
- The FroNtier/Squid solution has a rather important issue on cached data consistency. Some of the existing solutions were discussed on section 3.2.2. Generally, depending on the chosen cache invalidation policy, if cached data is used, FroNtier/Squid setups are very effective and can efficiently distribute data; conversely, if the data consistency mechanism is too strict, than cache contents will not be used and FroNtier/Squid solution becomes an inefficient data distribution mechanism;
- Application data access patterns are also very important as the query repetition rate will influence the efficiency of the solution. HEP applications do typically repeat the same queries and experts state that in the best cases, only 10% of the queries will not be cached while all the other 90% of the queries will be cached. This will only be validated at production phase.

At a bottom line, FroNtier is the primary replication technology used by the CMS experiment to distribute database contents to the different grid sites and is being tested by ATLAS as an alternative technology.

Oracle Streams

Oracle Streams is an alternative database distribution technique that is being deployed and tested on the LCG sites (see section 3.1). It is the base technology for database distribution for the experiments ATLAS and LHCb. Moreover, Oracle Streams will most probably be used by all major experiments to move the database data from the online site (the experiment pit) to the Tier-0 central grid site.

Although Oracle Streams allows multi-master replication (where one can write in different replicated databases), in the LCG 3D project data at tier-1 is considered to be read-only so that the complex deployment of multi-master replication is avoided.

Several Oracle Streams testing activities are under way within the LCG 3D project, namely, Oracle database replication tests from Tier-0 at CERN to all other Tier-1 grid sites, online-offline replication (replication from the experiment pit to the computer center at CERN), Oracle Streams monitoring framework, etc.

These testing activities presented different results that we can generally sum up:

- Oracle Streams is not a easy product to deploy and maintain; database experts are needed in both replication points and Oracle Streams experts are needed to install and maintain the setups;
- Oracle Streams replication processes are effective but comparatively slow; database replication times are very slow if compared to client access times;
- Although being a commercial product, Oracle Stream is not a mature product, specially if we consider the large scale of the problem;
- Oracle Streams solution benefits from the support Oracle Corporation gives to the LCG 3D project in terms of research staff (expertise);

- Having an Oracle Database deployed at each tier-1 site, although being a rather big deployment effort, represents a considerable boost in the system global performance, as around 10 Oracle databases will be servicing the same data;
- Oracle Streams is a reliable solution for database replication.

Comparison

As seen before, in HEP applications most data is read-only and the access patterns are repetitive. This is a base fact for the selection of database replication techniques as it defines that multi-master replication techniques are not needed and read-only replication techniques should be selected.

The two alternatives presented here, Oracle Streams and FroNtier/Squid, are not directly comparable as its working concepts are completely different. The LCG 3D project is working on both by deploying both Oracle databases and Squid servers in all Tier-1s. The basic objective of the project is to deploy and validate both database distribution technologies. As seen in previous sections, the two technologies have different advantages and disadvantages that should help decision makers decide what database distribution technology best fits their needs.

Oracle Streams, the heavy-weight commercial solution for database replication is, at a bottom line, a reliable solution. Nevertheless, it needs Oracle databases and Oracle Streams itself deployed and maintained. Although, Oracle Database services are already setup in some sites and the maintenance of these databases are almost not a trouble; other sites, have no Oracle databases and this solution implies new hires. By the other hand, Streams experts are a rare resource that will have to be centralized at CERN offering support to all grid sites.

Frontier, on its side, has still to prove its reliability and performance. Despite being very attractive as a light-weight solution for database distribution (no need for Oracle databases at Tier-1s or higher), cache consistency issues and lack of real scenario performance proofs (including integration with the complete software stacks) are still setting stakeholders aside.

6.2 Contributions

The major contributions of this work to the related research areas were:

- Synthesis of the state of art on grid technologies (see chapter Chapter 2 -);
- Detailed description of FroNtier and a brief description of existing database distribution techniques, including Oracle Streams (see chapter Chapter 3 -);
- Application of Jain's [70] methodology to a very practical case of performance testing of a distributed software package (see chapter Chapter 4 -);
- Detailed survey on existing Linux monitoring tools (see section 4.10.2);
- Development of a Linux based performance test framework (re-used for Oracle Database performance testing at CERN) (see section 4.10);
- Systematic FroNtier performance testing and detailed analysis.

6.3 Future Work

In this section we list some directions for future work:

- Test cases extension:
    - Execute the same test cases from a tier-1 site with squid servers at tier-1;
    - Execute the same test cases against several FroNtier servers to test the scalability of the system;
    - Use other workloads or software using CORAL, like CMS software;
    - Execute the same test cases with real data, when available;
    - Execute the same test cases against an Oracle database with a disabled cache, this would allow to measure the impact of the database cache on the obtained results;
    - Execute the same test cases against the production setup to test the real available capacity;
    - Execute the same test cases on the grid. Grid job synchronization problems would have to be handled;
- Statistical analysis of the test results obtained would show much more precise values and a statistical model describing the data taken in the experiments would allow the estimation of the contribution of each factor to the performance, the isolation of measurement errors, the estimation of confidence intervals for model parameters, etc;
- Extension of the test framework: automation of the statistical analysis and regression modeling, support for different clients, etc;
- Develop and test a cache invalidation prototype (see section 3.2.2.3); compare FroNtier with this cache invalidation mechanism with the results obtained on this work.

Acronyms

ACL	Access Control List
AFS	Andrew File System
ALICE	A Large Ion Collider Experiment (LHC experiment)
ARDA	A Realisation of Distributed Analysis for LHC
ATLAS	A Toroidal LHC ApparatuS (LHC experiment)
CA	Certificate Authority
CASTOR	CERN Advanced STORage Manager
CE	Computing Element: a Grid-enabled computing resource
CERN	European Organization for Nuclear Research
CMS	Compact Muon Solenoid (LHC experiment)
CPU	Central Processing Unit
DAQ	Data AcQuisition System
dCache	Hierachical storage manager (DESY, FNAL)
DGAS	DataGrid Accounting System
DPM	Disk Pool Manager
EDG	European Data Grid Project
EGEE	Enabling Grids for E-sciencE
ELFms	Extremely Large Fabric management system
FiReMan	File and Replica Catalogue
FTS	File Transfer Service
GB	Gigabyte
gLite	Lightweight middleware for Grid computing
Glue	Grid Laboratory Uniform Environment
GridFTP	Grid Service for File Transfer
GSI	Grid Security Infrastructure
HEP	High Energy Physics
HLT	High-Level Trigger
HPSS	High Performance Storage System
HTTP	HyperText Transfer Protocol
ICMP	Internet Control Message Protocol
I/O	Input/Output
IP	Internet Protocol
LCG	Worldwide LHC Computing Grid
LCG3D	LCG Distributed Deployment of Databases
LDAP	Lightweight Directory Access Protocol
LEAF	LHC-Era Automated Fabric
LEMON	LHC Era Monitoring
LCR	Logical Change Records
LFC	LCG File Catalogue
LFN	Logical File Name
LHC	Large Hadron Collider

LHCb	Large Hadron Collider beauty (LHC experiment)
LRMS	Local Resource Management System
LSF	Load Sharing Facility
MB	Megabyte
MIB	Management Information Base
MSS	Mass Storage System
MTU	Maximum Transmission Unit
PB	Petabyte (10^15 bytes)
PKI	Public Key Infrastructure
POSIX	Portable Operating System Interface
RAID	Redundant Array of Independent Disks
CORAL	COmmon Relational Access Layer
RDBMS	Relational Database Management System
RFIO	Remote File I/O
R-GMA	Relational Grid Monitoring Architecture
RLS	Replica Location Service
SE	Storage Element
SNMP	Simple Network Management Protocol
SQL	Structured Query Language
SQLite	SQL database engine
SRM	Storage Resource Manager
TB	Terabytes
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
VDT	Virtual Data Toolkit
VO	Virtual Organization
VOMS	Virtual Organization Management System
WMS	Workload Management System
XML	eXtensible Markup Language

References

[1] European Organization for Nuclear Research (CERN), http://www.cern.ch
[2] The Large Hadron Collider (LHC), http://www.cern.ch/lhc
[3] The LHC Computing grid (LCG) project, http://www.cern.ch/lcg
[4] J. Knobloch et al, LHC Computing Grid Technical Design Report, CERN-TDR-01 (http://lcg.web.cern.ch/LCG/tdr/), CERN, June 2005
[5] The Distributed Deployment of Databases for LCG (LCG3D) project, http://lcg3d.cern.ch/
[6] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the Grid: Enabling scalable virtual organizations. Lecture Notes in Computer Science, 2150, 2001
[7] I. Foster, C. Kesselman, J. Nick, and S. Tuecke, The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration, 2002
[8] I. Bird et al, LCG Baseline Services Group Report, June 2005
[9] EGEE Web Page, http://www.eu-egee.org/
[10] Open Science Grid (OSG) Web Page, http://www.opensciencegrid.org/
[11] Nordic Data Grid Facility (NDGF), http://www.ndgf.org/
[12] LCG-2 User Guide, https://edms.cern.ch/file/454439//LCG-2-UserGuide.html
[13] gLite Web Page, http://glite.web.cern.ch/glite/
[14] Laure E. et al, EGEE Middleware Architecture, 2005
[15] Burke S., Campana S., Peris A., et al, gLite 3.0 User Guide, Manual Series, 2006
[16] Globus Alliance, http://www.globus.org/
[17] Globus Toolkit Version 4: Software for Service-Oriented Systems. I. Foster. IFIP International Conference on Network and Parallel Computing, Springer-Verlag LNCS 3779, pp 2-13, 2005
[18] Virtual Data Toolkit (VDT), http://vdt.cs.wisc.edu/
[19] The Open Grid Services Architecture, Version 1.5, http://www.ggf.org/documents/GFD.80.pdf
[20] Open Grid Forum Web Site, http://www.ogf.org/
[21] Open Grid Services Infrastructure, http://www.ggf.org/documents/GFD.15.pdf
[22] Job Submission Description Language (JSDL), http://www.gridforum.org/documents/GFD.56.pdf
[23] gLite Workload Management Service User's Guide, https://edms.cern.ch/document/572489/1
[24] W. Allcock et al. GridFTP Protocol Specification. Global Grid Forum Recommendation GFD.20, March 2003.
[25] LCG Disk Pool Manager (DPM) administrator's guide, https://uimon.cern.ch/twiki/bin/view/LCG/DpmAdminGuide
[26] CERN Advanced Storage Manager (CASTOR), http://castor.web.cern.ch/castor/
[27] dCache, the commodity cache, 12th NASA Goddard and 21st IEEE Conference on Mass Storage Systems and Technologies, Spring 2004, Washington DC, USA
[28] High Performance Storage System (HPSS) Web Page, http://www.hpss-collaboration.org/

[29] The IBM Tivoli Framework Web Page, http://www-306.ibm.com/software/tivoli/
[30] EGEE gLite User's Guide, GLITE I/O, https://edms.cern.ch/file/570771/1.1/
[31] The gLite File Transfer System (FTS), https://twiki.cern.ch/twiki/bin/view/EGEE/FTS
[32] The File and Replica Manager (FiReMan) catalog user's guide, https://edms.cern.ch/file/570780
[33] S. Andreozzi, S. Burke, L. Field, S. Fisher, B. Kónya, M. Mambelli, J. M. Schopf, M. Viljoen, and A. Wilson, GLUE Schema Specification - Version 1.2, 2005
[34] The Relational Grid Monitoring Architecture (R-GMA), http://www.r-gma.org/
[35] The gLite Service Discovery User Guide, http://edms.cern.ch/document/578147
[36] R. Byrom et al., APEL: An implementation of Grid accounting using R-GMA, UK e-Science All Hands Conference, Nottingham, September 2005
[37] The gLite Accounting System (DGAS), http://www.cern.ch/glite/dgas/
[38] gLite Logging and Bookkeeping Service User's Guide, http://glite.web.cern.ch/glite/lb
[39] The gLite Job Provenance Service User Guide, http://egee.cesnet.cz/en/JRA1/JP-users-guide.pdf
[40] V. Welch et al, Security for Grid Services, Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing, 2003
[41] R. Housley, T. Polk, W. Ford and D. Solo, Internet X509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile, RFC3280 (2002)
[42] L. Pearlman, V. Welch, I. Foster, C. Kesselman, and S. Tuecke. A Community Authorization Service for Group Collaboration. In Proceedings of the IEEE 3rd International Workshop on Policies for Distributed Systems and Networks, 2002
[43] S. Farrel and R. Housley, An Internet Attribute Certificate Profile for Authorization, RFC3281, 2002
[44] R. Alfieri et al, VOMS, an Authorization System for Virtual Organizations. In Grid Computing, First European Across Grids Conference, 2004
[45] R. Alfieri et al, Managing Dynamic User Communities in a Grid of Autonomous Resources. CHEP03, California, USA, March 2003
[46] G. Carcassi et al., A Scalable Grid User Management System for Large Virtual Organization, proceedings of CHEP04, Interlaken, Switzerland, 2004
[47] S. Tuecke, V. Welch, D. Engert, L. Pearlman and M. Thompson, Internet X509 Public Key Infrastructure Proxy Certificate Profile, RFC3820, 2004
[48] J. Novotny, S. Tuecke, V. Welch, An Online Credential Repository for the Grid: MyProxy. Proceedings of the Tenth International Symposium on High Performance Distributed Computing (HPDC-10), IEEE Press, August, 2001
[49] Grid Operations Center, http://goc.grid-support.ac.uk/gridsite/gocmain/
[50] ELFms, Extremely Large Fabric management system, http://elfms.web.cern.ch/elfms/
[51] quattor, system administration toolsuite, http://www.quattor.org/
[52] LEMON — LHC Era Monitoring, http://lemon.web.cern.ch/lemon/
[53] LHC-Era Automated Fabric (LEAF), http://leaf.web.cern.ch/leaf/
[54] Oracle Database, http://www.oracle.com/technology/products/database/
[55] Oracle Materialized Views & Query Rewrite, May 2005, http://www.oracle.com

[56] Oracle Database Transportable Tablespace, http://www.oracle.com/technology/deploy/availability/htdocs/xtts.htm
[57] The SQLite RDBMS, http://www.sqlite.org/
[58] The MySQL RDBMS, http://www.mysql.com/
[59] Oracle Streams, http://www.oracle.com/technology/products/dataint/htdocs/streams_fo.html
[60] Lueking, L. et al, FroNtier: High Performance Database Access Using Standard Web Components in a Scalable Multi-tier Architecture, CHEP 04, 2004
[61] Squid web proxy cache, http://www.squid-cache.org/
[62] Duellmann D. et al, LCG 3D project status and production plans, CHEP 06, 2006
[63] Java Servlet Technology, http://java.sun.com/products/servlet/
[64] The Apache Tomcat servlet container, http://tomcat.apache.org/
[65] The Extensible Markup Language (XML), http://www.w3.org/XML/
[66] COmmon Relational Abstraction Layer, http://pool.cern.ch/coral/
[67] I. Papadopoulos et al, CORAL relational database access software, CHEP 06, 2006
[68] FroNtier Plug-in to CORAL, http://pool.cern.ch/coral/currentReleaseDoc/FroNtierAccess/
[69] Oracle Real Application Clusters, http://www.oracle.com/technology/products/database/clustering/
[70] Jain R., The art of computer systems performance analysis, Wiley, 1991
[71] Ezolt, P., Optimizing Linux Performance: A Hands-on Guide to Linux Performance Tools, Prentice Hall PTR, 2005
[72] Johnson S., Huizenga G., Pulavarty B., Performance Tuning for Linux Servers, IBM Press, 2005
[73] Uptime man page, http://www.linuxmanpages.com/
[74] N. J. Gunther, The Practical Performance Analyst, Authors Choice Press, ISBN: 0-59-512674-X, 2000
[75] Mpstat man page, http://www.linuxmanpages.com/
[76] Vmstat man page, http://www.linuxmanpages.com/
[77] Ps man page, http://www.linuxmanpages.com/
[78] Iostat man page, http://www.linuxmanpages.com/
[79] df man page, http://www.linuxmanpages.com/
[80] Tanenbaum, Andrew. Computer Networks, Third Edition. Prentice Hall, 1996
[81] netstat man page, http://www.linuxmanpages.com/
[82] Strace page, http://sourceforge.net/projects/strace

Appendix I – Test Web Report

This appendix presents an example of a web report generated by the test framework developed in this work. As seen in section 4.10.1, this web report contains detailed information about the test case it refers to and detailed monitoring plots of the servers involved in the test case.

The first section shows the details of the experiment:

SETUP

Test directory: /afs/cern.ch/user/l/lramos/work/perf/logs/athena17boxes_method_Squid_rep-1

Client test nodes: ['lxb0677', 'lxb0678', 'lxb0679', 'lxb0680', 'lxb0682', 'lxb0756', 'lxb0757', 'lxb0758', 'lxb0759', 'lxb0761', 'lxb0762', 'lxb0763', 'lxb0764', 'lxb0765', 'lxb0766', 'lxb0767', 'lxb0768']

Servers Under Test: ['lxb5556', 'lxfsrk402']

Number of test clients: 51 -> 17 node(s) running 3 client(s each)

Client executable: athenaEXEC.sh

Query List: queryList.xml

Test Configuration file: testConf.xml

Test Client Execution Script: runAthenaClient.sh

Here we can see the test directory and name (athena17boxes_method_Squid_rep-1), the list of client nodes, the names of the servers, the number of test clients (17 nodes running 3 clients each), and the browsable configuration files: the Query List file contains the clients' workload script (the list of queries all clients should execute); the Test Configuration file is client configuration script and contains the test parameters; and the Client Execution Script contains some more client specific test parameters.

After the setup section, the test execution time is printed for reference:

=>> Test run on from **28/Feb/2007 at 18:47 to 28/Feb/2007 at 20:17** <<=

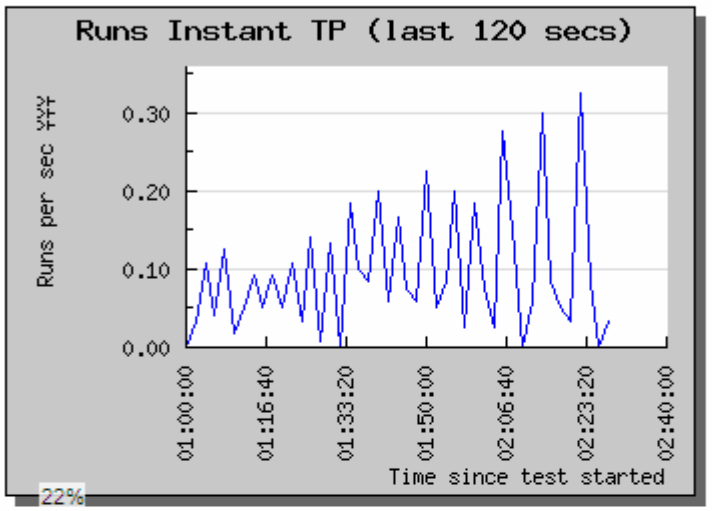
The results section contains the data gathered during the test execution:

RESULTS

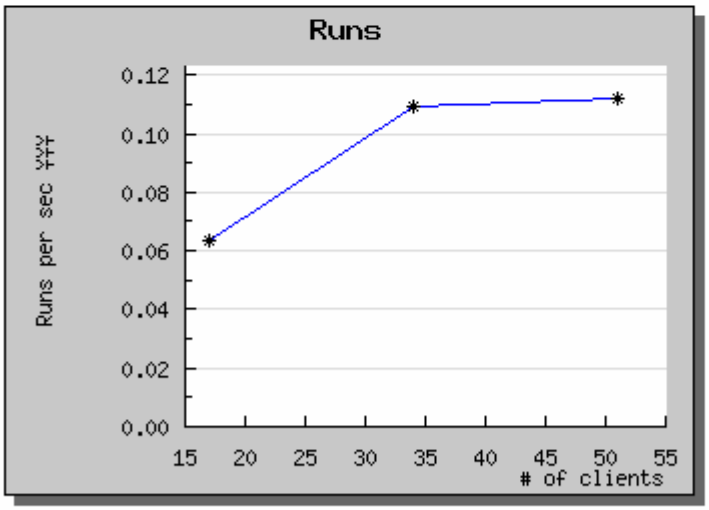
Client report rate: 100 percent (51 out of 51) (see individual client results)

Instant Throughput

per sec in the last 120 secs



Throughput per number of clients



# of clients	Time interval (secs)	Runs(#/sec)
17	1787	0.06379
34	1792	0.10938
51	1790	0.11173

The results section shows the following information:

- individual client results with execution time, number of queries, error reports, individual throughput, etc.;
- instant aggregated throughput (plot with the aggregated throughput every 2 minutes of the experiment);
- aggregated throughput per number of clients (plot with aggregated throughput for each ramp up stage with different number of clients);
- table with aggregated throughput values per number of clients.

The monitoring section (see below) contains monitoring information (collected from LEMON) about all the client nodes and servers, for example, the network and the CPU utilization. The next plots show the network utilization (write and read) and the CPU data (idle CPU% and load average) of a Frontier server machine (lxb5556) on a given test execution.